Usare gli Oggetti

- Modificatori di accesso
  - Pubblico, privato, protetto, default
  - Elementi di tipo statico
    e di tipo istanza
- Stringhe
- Array di oggetti

Oggetti e utilizzatori

- Non tutte le caratteristiche di un oggetto devono essere visibili dall'esterno
  - Rischio di manomissioni indebite
  - Occorre separare l'interfaccia dall'implementazione
- Si realizza l'incapsulamento utilizzando un modificatore di visibilità
  - Metodi e attributi possono preceduti da una parola chiave che indica il livello di privilegio loro associato

Modificatori di visibilità

- Private
  - Indica che si può accedere all'elemento solo da altri metodi appartenenti alla **stessa classe**
- Public
  - Indica che l'elemento può essere utilizzato da metodi di **qualsiasi classe**
- *Default]*
  - In assenza di indicazioni, l'elemento è accessibile alle altre classi dello **stesso "gruppo"** (package)
- *Protected*
  - Indica che l'elemento è accessibile alle sottoclassi della classe e alle altre classi dello stesso package

Modificatori di visibilità

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Modificatori di visibilità

- Esercizio
  - 1) Definire il package **A**
  - 2) Definire il package **B**
  - 3) Creare la classe **Persona** in **A**
    - **private String nomePrivate**
    - **default int cognomeDefault**
    - **protected String etaProtected**
    - **public String cellNumPublic**
  - 4) Creare la classe **TestVisibilitaDaA in A** (da stesso package)
  - 5) Creare la classe **TestVisibilitaDaB in B** (da altro package)
  - 6) Creare la classe **TestVisibilitaDaInheritedA** extends **Persona** in **A** (In stesso package)
  - 7) Creare la classe **TestVisibilitaDaInheritedB** extends **Persona** in **B** (da altro package)
  - 8) Definire il metodo void **TestVisibilita()** nelle classi di test create

Modificatori di visibilità

- Verifica visibilità in **TestVisibilita()** di nome, cognome, eta, cellNum nelle classi:
  - 9) **TestVisibilitaDaA in A** (da stesso package)
    - Person p = new Person();
    - p.nomePrivate Not **visibile**
    - p.cognomeDefault
    - p.etaProtected
    - p.cellNumPublic
  - 10) **TestVisibilitaDaB in B** (da altro package)
    - Person p = new Person();
    - p.nomePrivate Not **visibile**
    - p.cognomeDefault Not **visibile**
    - p.etaProtected Not **visibile**
    - p.cellNumPublic

Modificatori di visibilità

- Verifica visibilità in **TestVisibilita()** di nome, cognome, eta, cellNum nelle classi:
  - 10) **TestVisibilitaDaInheritedA** extends **Persona** in **A** (In stesso package)
    - Person p = new Person();
    - p.nomePrivate Not **visibile**
    - p.cognomeDefault
    - p.etaProtected
    - p.cellNumPublic
  - 11) **TestVisibilitaDaInheritedB** extends **Persona** in **B** (da altro package)
    - Person p = new Person();
    - p.nomePrivate Not **visibile**
    - p.cognomeDefault Not **visibile**
    - p.etaProtected
    - p.cellNumPublic

Visibilità delle classi

- Anche le classi possono essere precedute da un modificatore di visibilità
  - In questo caso, “private” non ha senso
- Se una classe è dichiarata pubblica, può essere utilizzata da classi appartenenti a qualsiasi gruppo (package)
  - Altrimenti, può essere usata solo nell’ambito del gruppo in cui è stata definita

Incapsulamento

- Per massimizzare il riuso, solo l'**informazione minima** necessaria al funzionamento deve essere accessibile
- Di solito:
  - Attributi **privati**
  - Metodi **pubblici**
  - Costruttori **pubblici**

Incapsulamento

```
public class Cerchio {
  private double r;
  public Cerchio() {
        ...
  }
  public void setRaggio(double val)
 {
        r=val;
  }
}
```

Metodi e attributi

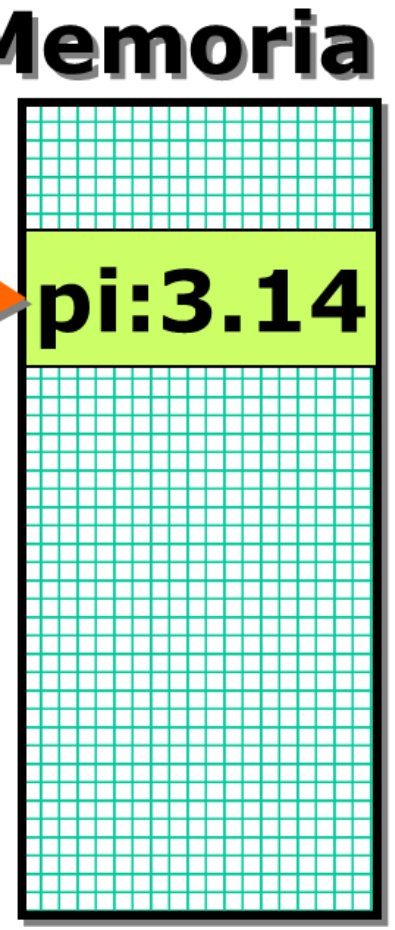
- Oggetti appartenenti ad una stessa classe hanno lo stesso insieme di attributi
  - Ogni oggetto ha però i propri valori, indipendenti dagli altri
  - Un metodo opera sui valori dell'oggetto su cui è invocato
  - L'evoluzione dello stato di un oggetto è indipendente da quella di altri oggetti della stessa classe

Attributi statici

- Talora, si vuole memorizzare un'informazione comune a tutti gli oggetti di una data classe
  - Si utilizzano gli attributi "statici"
  - Il loro valore viene conservato in un blocco di memoria separato, relativo alla classe
  - Sono analoghi a variabili globali in altri linguaggi (C, C++)

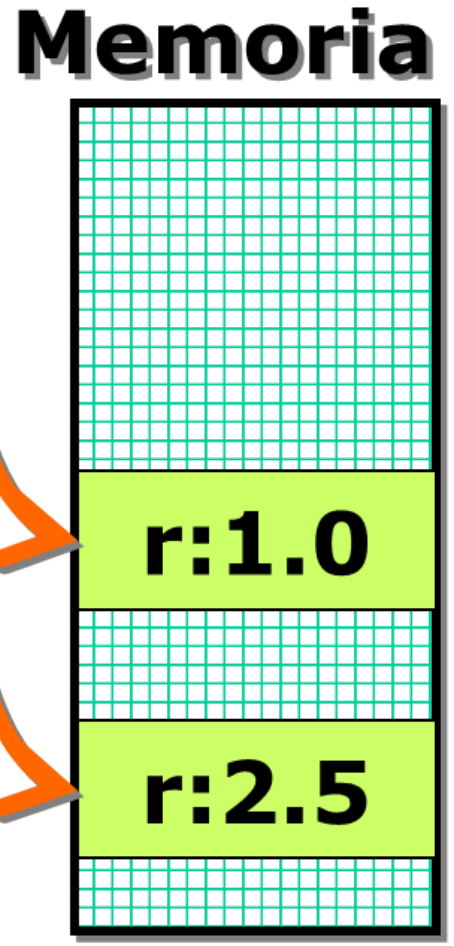
Esempio

```
class Cerchio {
  static double pi=3.14;
  double r;
...
}
```



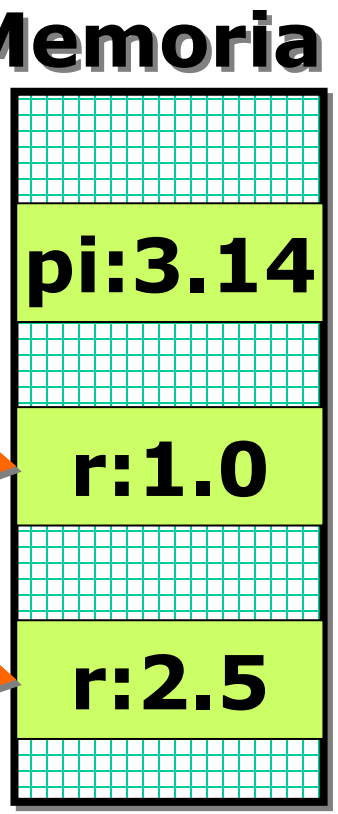
Esempio

```
Cerchio c1=
      new Cerchio(1.0);
Cerchio c2=
      new Cerchio(2.5);
```



Esempio

```
Cerchio c1=
      new Cerchio(1.0);
Cerchio c2=
      new Cerchio(2.5);
```



Esempio

```
class Cerchio {
  ...
    double calcolaArea() {
       double a;
       a = r*r*pi;
       return a;
  }
}
```

Uso degli attributi statici

- Si può fare accesso ad un attributo statico anche in mancanza di un oggetto specifico
  - double d= Cerchio.pi/2;
- Gli attributi statici devono essere inizializzati
  - Non si può fare nel costruttore!
  - Occorre farlo all'atto della dichiarazione
  - static double pi = 3.14 ;

Esercizio incapsulamento+JUnit

- Creare classe Class con double raggio e static Double pi=3.14
- Creare metodo calcolaArea, toString(), getter e setter
- Creare classe Junit CerchioTest
- Istanziare due oggetti Cerchio
  - Test raggio cerchio1 e cerchio2
  - Test che pi sia sempre lo stesso valore condiviso
  - Calcolare l'area e verifica che il calcolo sia corretto
    - Sfruttare BigDecimal per arrotondamento decimali double
    - Test eguaglianza fra stringhe valori aspettati e calcolati

Esercizio incapsulamento+JUnit

```java
package incapsulamento;

public class Cerchio {
    private double raggio;     // Modificabile solo da setRaggio()
                               // Lggibile solo con getRaggio()
                               // Inizializzabile solo dal costruttore
    static Double pi=3.144;
    public Cerchio(double raggio) {
        this.raggio = raggio;
    }
    public void setRaggio(double raggio) {
        this.raggio = raggio;
    }
    public double getRaggio() {
        return this.raggio;
    }
    double calcolaArea() {
        return raggio*raggio*pi;
    }
    @Override
    public String toString() {
        return "Cerchio [raggio=" + raggio + "]";
    }
}
```

Esercizio incapsulamento+JUnit

```java
package incapsulamento;
import static org.junit.jupiter.api.Assertions.*;
import java.math.BigDecimal;
import java.math.RoundingMode;
import org.junit.jupiter.api.Test;
class CerchioTest {
    @Test
    void test() {
        Cerchio c1 = new Cerchio(10);
        Cerchio c2 = new Cerchio(20);
        int a[] = {1,2,3,4,5};
        for (int i : a) {
            System.out.println("a="+i);
        }
        assertTrue(c1.getRaggio() == 10);
        assertTrue(c2.getRaggio() == 20);
        assertTrue(c1.pi == c2.pi);    // pi dovrebbe essere acceduto in modo statico

        System.out.println("Pigreco="+Cerchio.pi);
        double y = c1.calcolaArea();
        BigDecimal ybd = BigDecimal.valueOf(y).setScale(1, RoundingMode.DOWN);
        BigDecimal ybd3144 = BigDecimal.valueOf(Cerchio.pi * 100).setScale(1, RoundingMode.DOWN);

        assertTrue(ybd.toString().equals(ybd3144.toString()));
    }

}
```

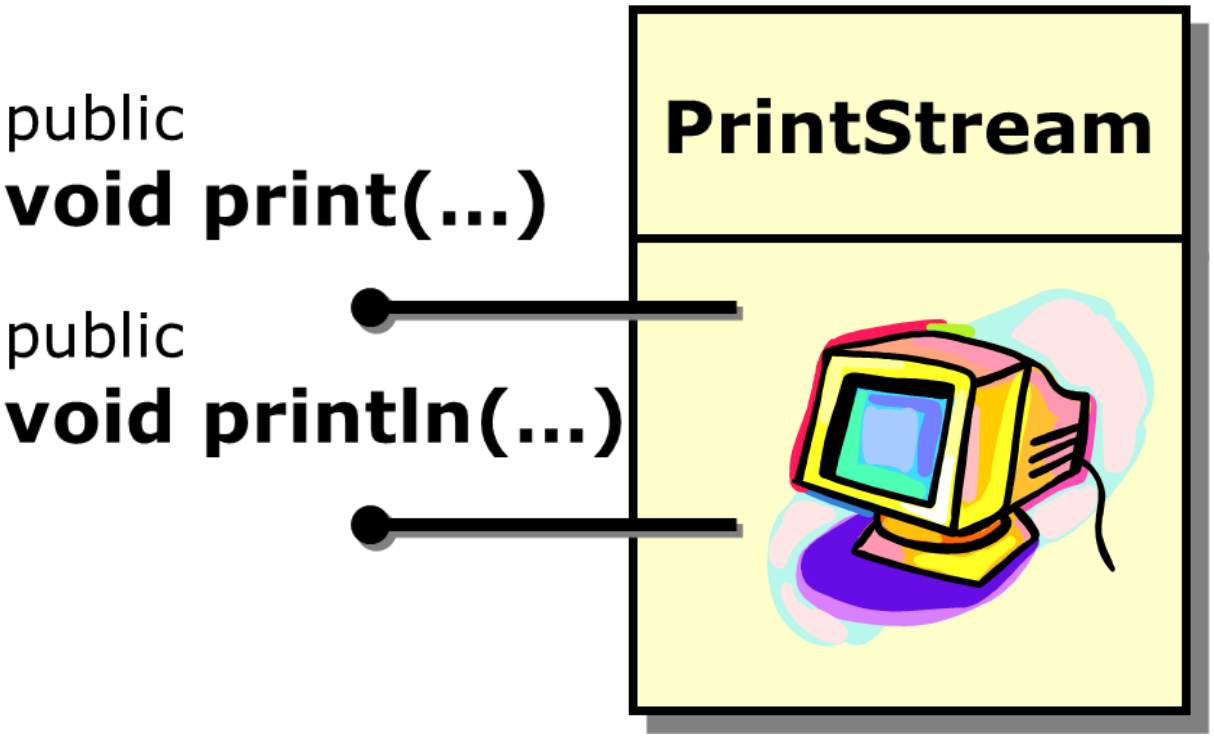
Metodi statici

- Non fanno riferimento a nessun attributo specifico di un oggetto
  - Preceduti dal modificatore “static”
  - Equivalenti a procedure e funzioni di altri linguaggi
  - Possono essere invocati a partire dal nome della classe

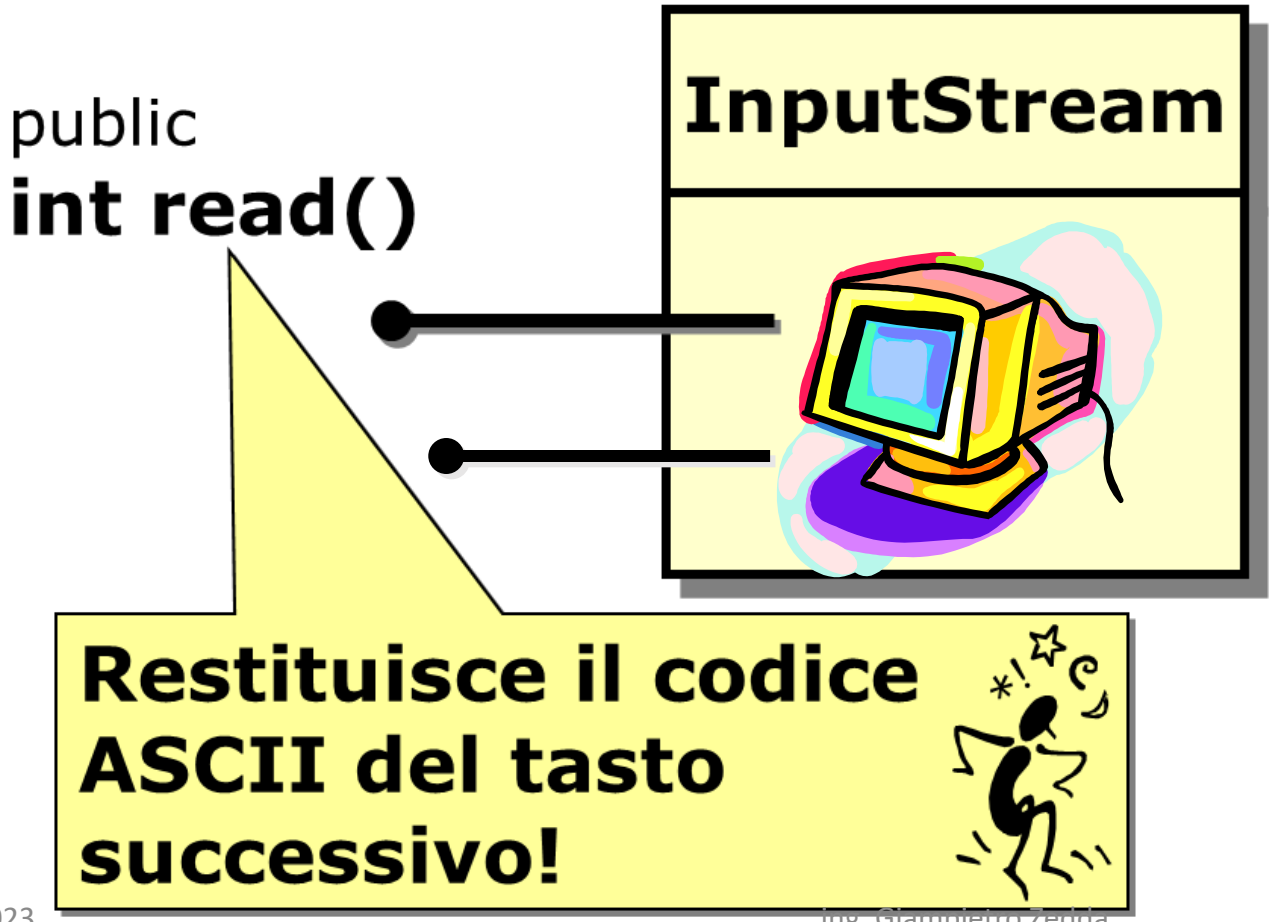
Esempi

- La classe “Math” contiene metodi statici per le principali operazioni matematiche
  - double d1,d2;
  - d1 = **Math.sqrt**(2.0);
  - d2 = **Math.sin**(**Math.PI** / 2.0);
- La classe “System” contiene, sotto forma di attributi statici, oggetti che modellano le interazioni con la console di sistema
  - **System.out** //output su schermo
  - **System.in** //input da tastiera

System.out



System.in


public
int read()
InputStream
Restituisce il codice ASCII del tasto successivo!

Il metodo main()

- Quando un programma inizia, non può esistere ancora nessun oggetto
  - Il punto di ingresso deve essere un metodo statico e pubblico
  - Perché?
- Convenzione Java:
  - **public static void main(**String[ ] args**)**

Esempio

```
public class Hello {
  public static void
      main(String[] args) {
    System.out.println(“Hello Java”);
  }
}
```

La classe String

- Classe che modella **sequenze immutabili** di caratteri
- Sintassi semplificata
  - **String s1= “Hello” ;**
  - **String s2 = s1+ “ Java”**
- Offre molti metodi
  - Confronto, ricerca, derivazione di nuove stringhe, informazioni generali, …

Confronto e ricerca

- public boolean **equals**(String s)
  - Restituisce true se il parametro contiene gli stessi caratteri dell'oggetto corrente
- public boolean **equalsIgnoreCase**(String s)
  - Idem, trascurando la differenza tra maiuscole e minuscole
- public int **indexOf**(String s)
  - Restituisce la posizione, all'interno della sequenza di caratteri, in cui inizia la stringa indicata come parametro (-1 se non esiste)

Derivazione e informazioni

- public String **toUpperCase**()
  - Restituisce una **nuova** stringa contenente gli stessi caratteri in versione maiuscola
- public String **replace**(char oldChar, char newChar)
  - Restituisce una **nuova** stringa in cui tutte le occorrenze di *oldChar* sono sostituite con *newChar*
- public String **substring**(int beginIndex, int endIndex)
  - Restituisce una **nuova** stringa formata dai caratteri che iniziano alla posizione *beginIndex* fino a *endIndex* escluso
- public int **length**()
  - Restituisce la lunghezza in caratteri della stringa corrente

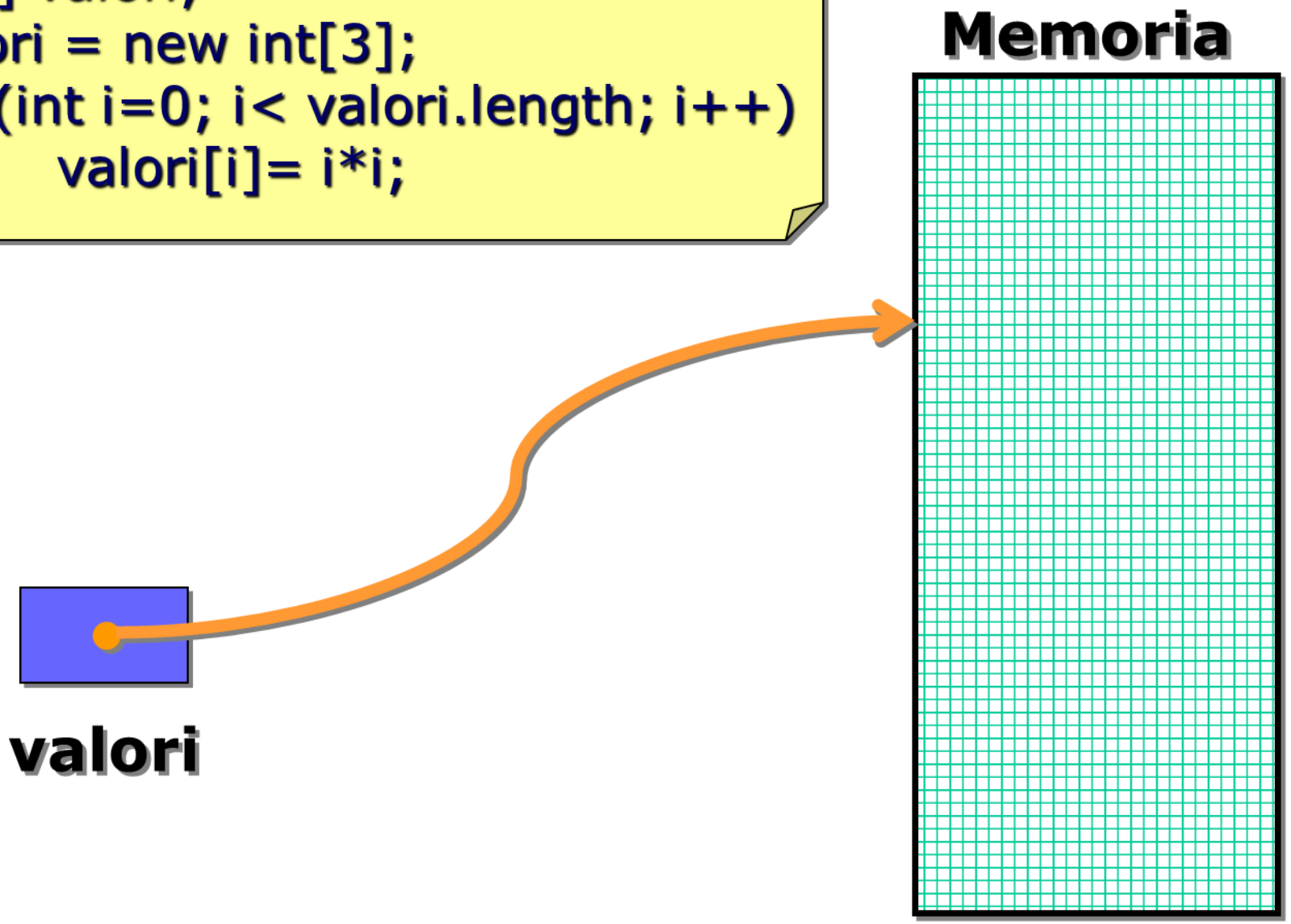
Esempi

```
String s1="ciao";
String s2= s1.toUpperCase();
boolean b= s2.equals (s1);
int i= s2.length();
int j= s2.indexOf("AO");
String s3=s1.substring(j,i);
char c=s1.charAt(0);
```

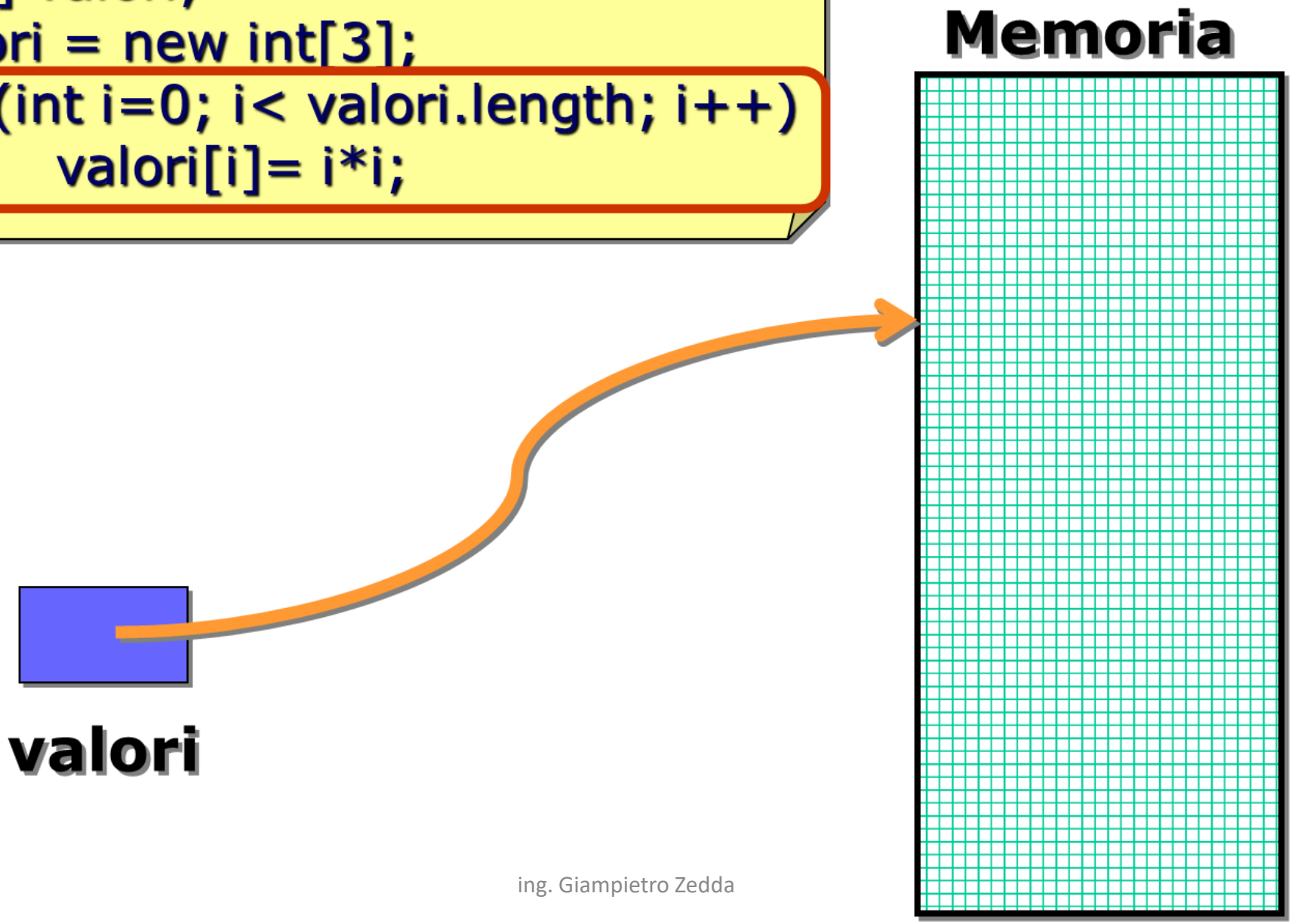
Array

- All'interno di un programma può essere necessario gestire **collezioni** di dati:
  - Il modo più semplice per organizzarli, se sono di tipo omogeneo, è attraverso il concetto di **array** o vettore
- Array in Java: oggetti che incapsulano **sequenze ordinate** di dati
  - Hanno dimensione **fissa**, definita all'atto della **creazione**
  - Sintassi **semplificata** per accedere ai singoli elementi

```
int[] valori;
valori = new int[3];
for (int i=0; i< valori.length; i++)
        valori[i]= i*i;
```



```
int[] valori;
valori = new int[3];
for (int i=0; i< valori.length; i++)
        valori[i]= i*i;
```



Tipologie di array

- Esistono due gruppi di array
  - Quelli che modellano sequenze di **tipi elementari** (interi, reali, caratteri, booleani)
  - Quelli che modellano sequenze di **oggetti**
- Qualunque sia il gruppo di appartenenza, gli array sono sempre oggetti **complessi**
  - Vengono creati tramite l'operatore **new**
  - Si accede ad essi attraverso un riferimento in memoria

Inizializzare un array

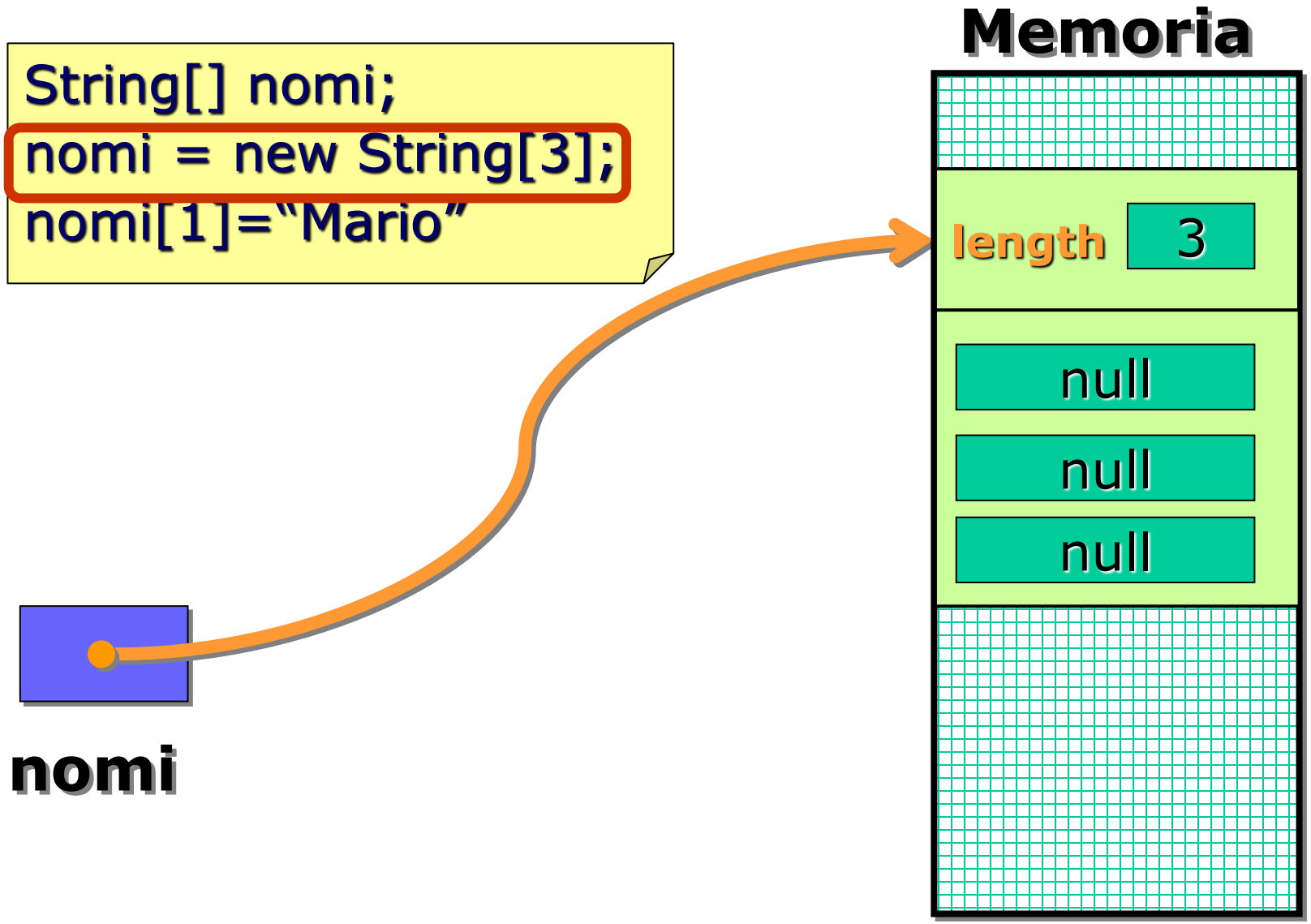
- Nel caso di array di tipi elementari, all'atto della creazione vengono assegnati valori di default
  - ***0*** per i numeri (interi e reali)
  - ***false*** per i valori logici
  - ***'\000'*** per i caratteri
- A volte si conosce a priori l'intero contenuto dell'array
  - Java offre una sintassi semplificata per la sua inizializzazione

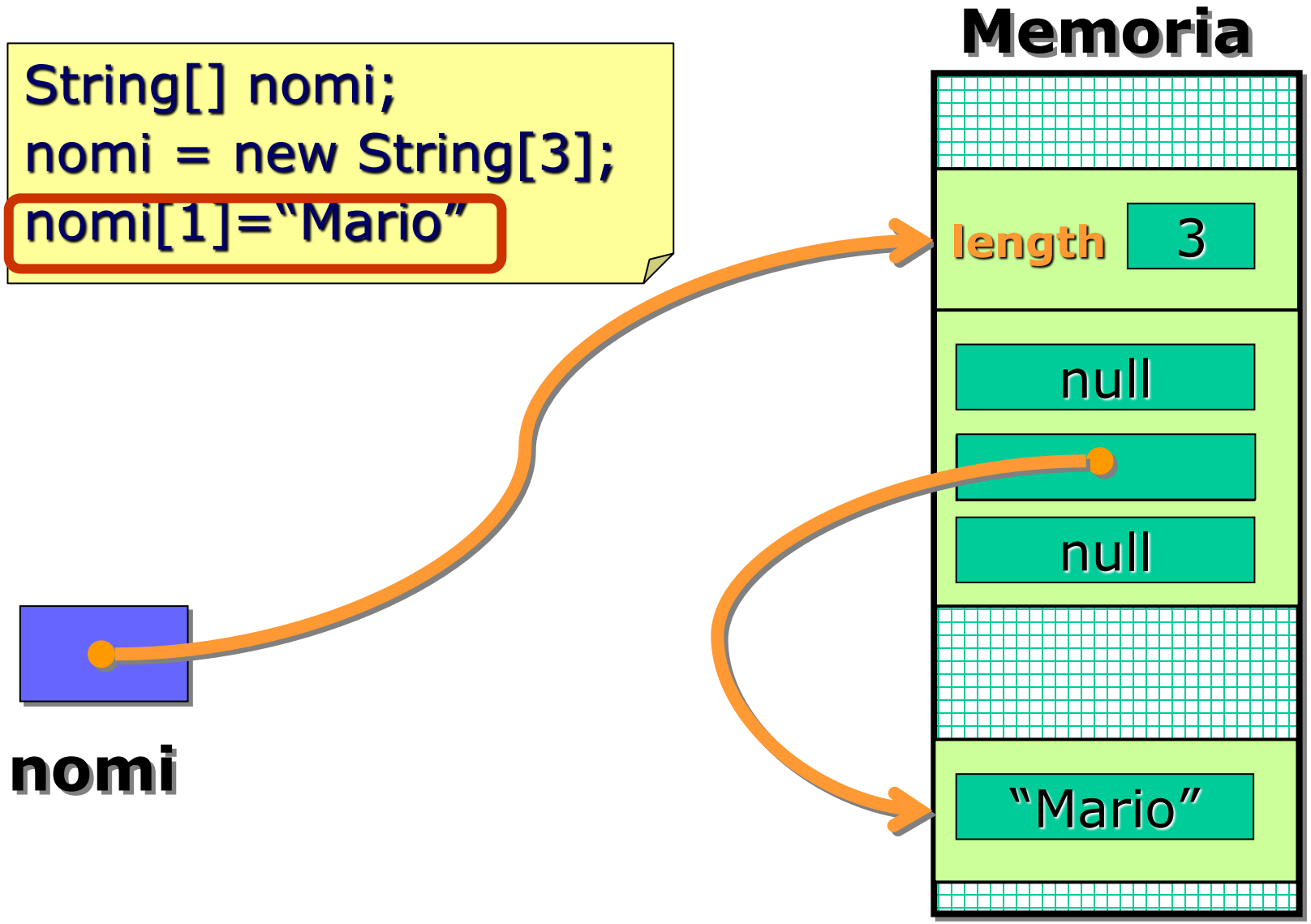
```
int[] valori = new int[] {4, 0, 7, 9};
```

Uso degli array

- Per accedere ad un singolo elemento, si utilizza la sintassi
  - ***nome_array [indice]***
  - L'indice deve essere compreso tra 0 e il numero di elementi presenti nell'array
- Un array può contenere anche tipi complessi (oggetti)
  - In questo caso, nelle singole posizioni sono memorizzati i riferimenti agli oggetti specifici
  - All'atto della creazione del vettore, tutti i riferimenti contenuti valgono **null**

```
String[] nomi=
        new String[10];
```


String[] nomi;
nomi = new String[3];
nomi[1]="Mario"
Memoria
length
3
null
null
null
nomi


String[] nomi;
nomi = new String[3];
nomi[1]="Mario"
Memoria
length
3
null
null
"Mario"
nomi

Array a più dimensioni

- È anche possibile creare matrici
  - int [ ] [ ] matrice= new int [3] [3];
- Si tratta di “array di array”
  - Il primo indice si riferisce alla riga, il secondo alla colonna
- Non c’è limite al numero di dimensioni dell’array
  - Se non la memoria disponibile…

```
int[][] matrice = new int[3][3];
matrice[2][0]= 1;
```