

Array (continua)

Tipo [] nome = new Tipo [dimensione];

float[] a1 = new float [5];

int l = a1.length; // l vale 5

a2 = new float [10];

l = a2.length; // l vale 10

Non esiste arithmetica di puntatori

a1 ++ // errore

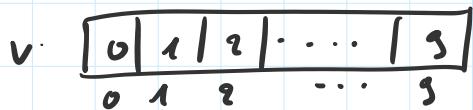
l'operatore di selezione [] controlla
che non vengano sfiorati i limiti dell'array

$0 \leq \text{indice} < \text{dimensione array}$

int[] v1 = new int [5];

v1[7] ... // errore: lancer
// ArrayIndexOutOfBoundsException

```
int[] v = new int[10];  
for (int i=0; i<v.length; i++)  
    v[i] = i;
```



Iinizializzazione

Valori di default

- false per boolean
- 0 per tipi numerici
- null per riferimenti

```
boolean[] v2 = new boolean[1000];
```

è possibile fornire valori iniziali

```
int[] v3 = { 50, 100, 150 };
```

```
int[] v4 = new int[] { 5, 4, 3, 2, 1 };
```

Array di tipo riferimento

Supponiamo di avere

public class Studente {

private String nome;

private String cognome;

private double media;

public Studente (String n, String c) {

nome = n;

cognome = c;

}

public double getMedia () {

return media;

}

:

}

se scrivo

Studente [] ss = new Studente [5];

creo un array di 5 elementi

ogni elemento è un riferimento a Studente



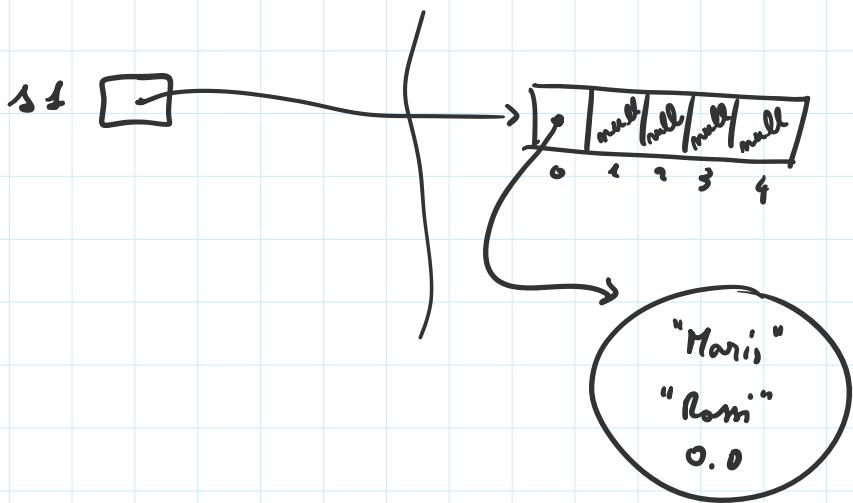
non ci sono oggetti del tipo Studente

tutti gli elementi di ss sono inizializzati
a null

double m = ss[0].getMedia(); //errore!

↑
genera NullPointer Exception

`s1[0] = new Studente("Mario", "Rossi");`

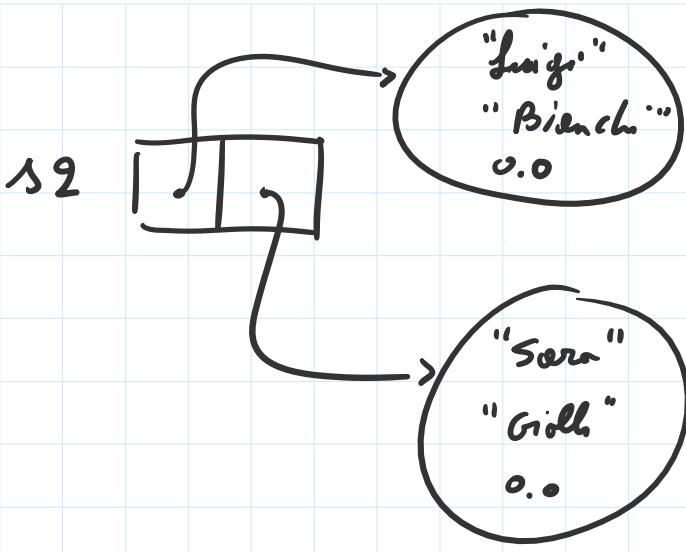


inizializzazione

`Studente[] s2 = { new Studente("Luigi", "Bianchi"),
new Studente("Sara", "Gialli")};`

oppure

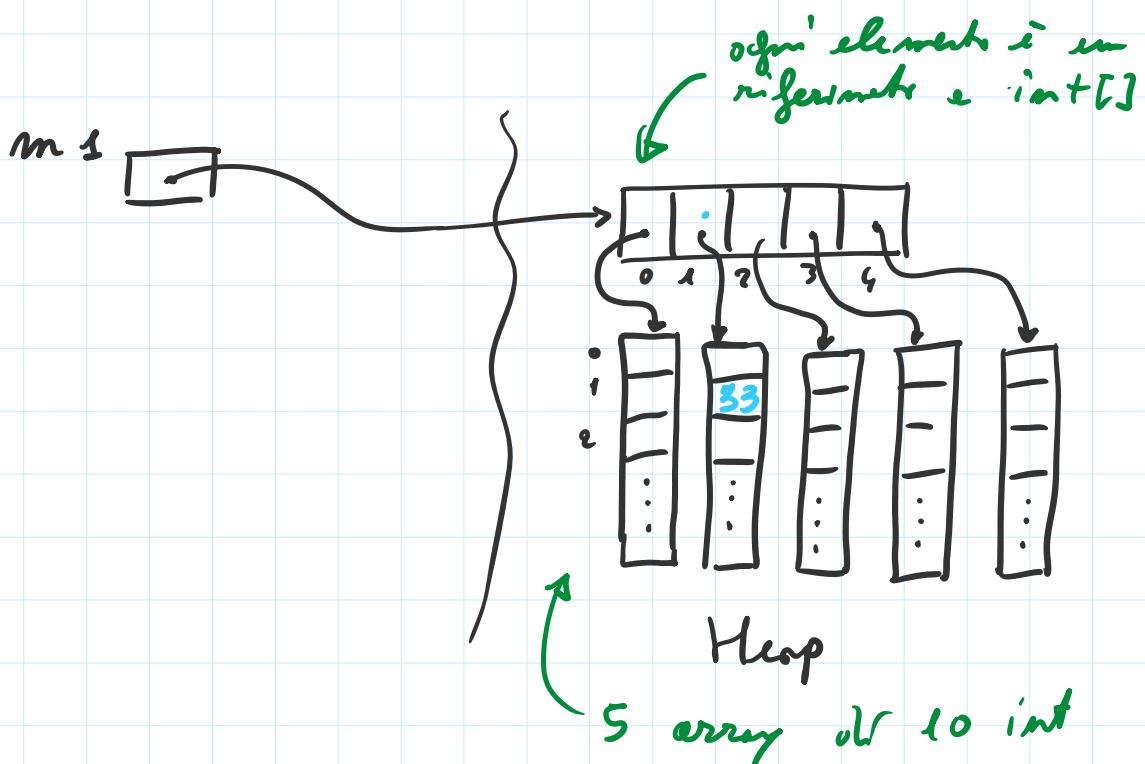
`Studente[] s3 = new Studente[] {
new Studente(" -. ", "- "),
new Studente(" -. ", "... "),
new Studente(" -. ", "- . ")};`



Array multidimensionali

Per creare un array multidimensionale

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



```
ms[1][1] = 33;
```

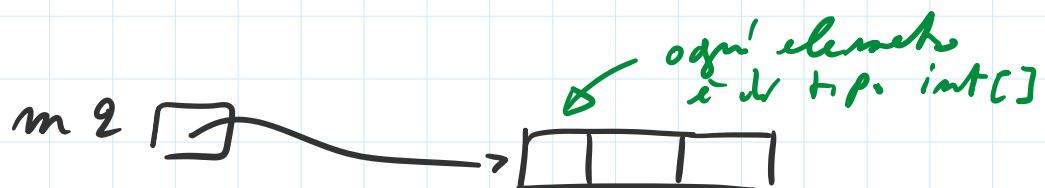
```

for (int i=0; i<m1.length; i++)
    for (int j=0; j<m1[i].length; j++)
        m2[i][j] = ...

```

L'ultima dimensione puo' essere non specificata

`int[][] m2 = new int[3][];`

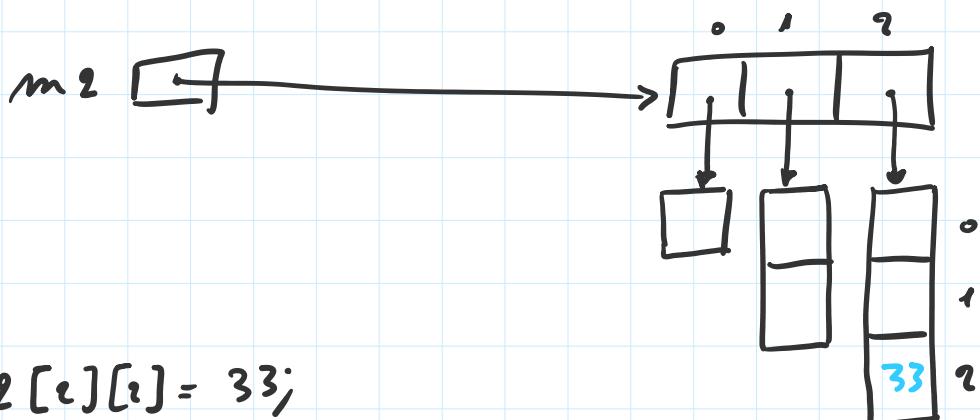


`m2[0][0] = 33;` // errore

```

for (int i=0; i<m2.length; i++)
    m2[i] = new int[i+1];

```



Alcuni metodi che possono essere utili

[https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#arraycopy\(java.lang.Object,%20int,%20java.lang.Object,%20int,%20int\)](https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#arraycopy(java.lang.Object,%20int,%20java.lang.Object,%20int,%20int))

```
public static void arraycopy(Object src,
    int srcPos,
    Object dest,
    int destPos,
    int length)

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array..
```

Metodo della classe System

per chiamarlo System.arraycopy(...)

La classe Arrays

<https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

mette a disposizione alcuni metodi utili a ordinare array, cercare valori in un array, etc

Per esempio:

binarySearch

```
public static int binarySearch(int[] a,
    int key)
```

Searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted (as by the sort(int[]) method) prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

Parameters:

- a - the array to be searched
- key - the value to be searched for

oppure

sort

```
public static void sort(double[] a)
```

Sorts the specified array into ascending numerical order.

The < relation does not provide a total order on all double values: $-0.0d == 0.0d$ is true and a Double.NaN value compares neither less than, greater than, nor equal to any value, even itself. This method uses the total order imposed by the method Double.compareTo(java.lang.Double): $-0.0d$ is treated as less than value $0.0d$ and Double.NaN is considered greater than any other value and all Double.NaN values are considered equal.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers $O(n \log(n))$ performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

- a - the array to be sorted

Stringhe

Istanzie della classe String

Non sono array di caratteri terminati da '\0'

Sono immutabili

Per "stringhe" modificabili c' sono le classi

le class'

String Buffer

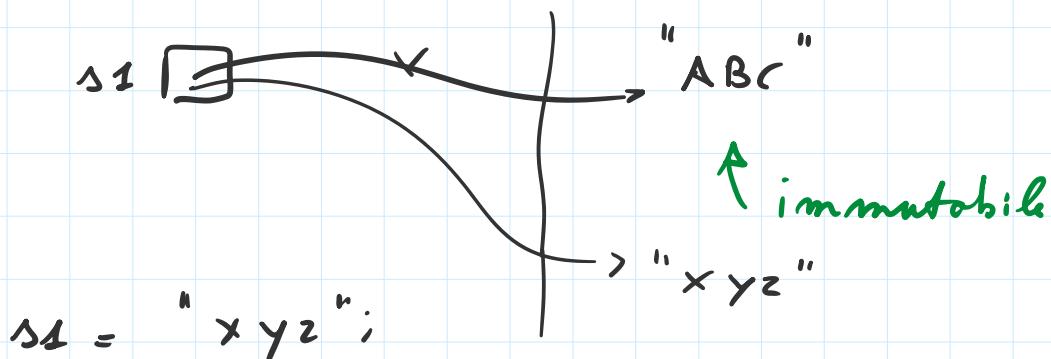
è thread-safe, può essere
manipolata da più thread

String Builder

dove essere manipolata
da un solo thread

Per creare stringhe

String ss = "ABC";



per concatenare stringhe : +

String ss = "ciao";

String s3 = "mondo";

String s4 = ss + " " + s3;

// s4 vale "Ciao mondo"

+ può essere applicato a stringhe
e valori di altri tipi

+ può essere applicato a stringhe
e valori di altro tipo

int v = 4;

String r = "Rocky";

String t = r + v; // t vale "Rocky4"

↑ R
String int viene convertito in String

int w = 5;

String y = r + v + w;

"Rocky4"
"Rocky45"

String y2 = v + w + r;

g
"gRocky"

Alcuni metodi utili

public String substring (int beginIndex,
int endIndex)

restituisce la sotto-stringa tra

String sc = "Smiles";
String sb = sc.substring(1, 5);
// sb vale "miles"

public int length() Note: è un metodo

restituisce la lunghezza di una stringa

String sc = "ABC";
int k = sc.length(); // k vale 3

se vuole conoscere il carattere i-esimo

public char charAt(int i)

esempio

char c = sc.charAt(1); // c vale 'B'

Per costruire una stringa a partire da un array di char:

char[] ar = new char[] {'c', 'i', 'a', 'o'}

String sol = new String(ar);

Confronto tra stringhe - importante -

Confronto tra stringhe - importante -

Per confrontare stringhe dobbiamo usare il metodo equals()

Non usare == per confrontare stringhe

Il metodo equals restituisce true se le stringhe che vengono confrontate hanno gli stessi caratteri (false altrimenti).

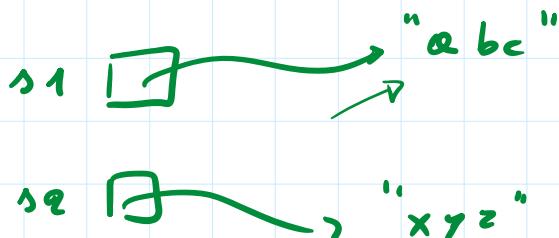
L'operatore == confronta i riferimenti e restituisce true se e solo se i due riferimenti puntano allo stesso oggetto.

String s1 = "abc";

String s2 = "xyz";

boolean b1 = s1.equals(s2); // b1 vale false

boolean b2 = s1 == s2; // b2 vale false



String s3;

s3 

s3 = "ab";

char x = 'c';

s3 = s3 + x; // s3 vale "abc"

b1 = s1.equals(s3); // true

b2 = s3 == s1; // ? potrebbe non essere true

String s4 = "abc";

Usare SEMPRE equals()

Metodo main

public static void main(String[] args)

↑
valori forniti
da riga di comando

public class Stampa {
 public static void main(String[] args) {
 for (int i = 0; i < args.length; i++)
 System.out.println(args[i]);
 }
}

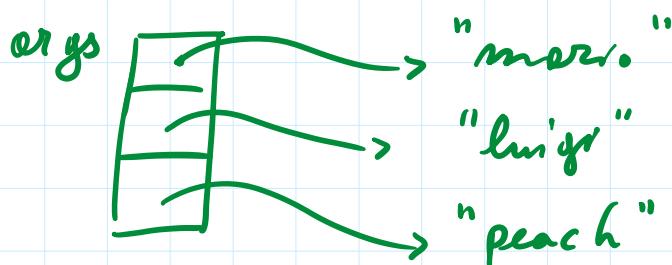
```

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

```

\$ java Stampo mero. luigi peach

↑
args[0] ↑
args[1] ↑
args[2]



Più classi possono avere il metodo main

Motivi:

2) main di prova

public class Stampante {
=}

public static void main(String[] args){
Stampante s = new Stampante();
// codice di prova della
// classe Stampante

} }

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

```
public class Principale {
    public static void main (String [] args) {
        // main verso e proprio
    }
}
```

- 2) l'applicazione può funzionare in
ambiti diversi ognuna identificata
da un nome

```
public class GUI {
    public static void main (String [] args) {
        ;
    }
}
```

```
public class CLI {
    public static void main (String [] args) {
        ;
    }
}
```

Ereditarietà

Ci permette di definire nuove classi a partire
da classi già esistenti

Le nuove classi sono specializzazioni delle classi già esistenti

Aiuta a rimuovere il codice

Si esprimono le sole differenze

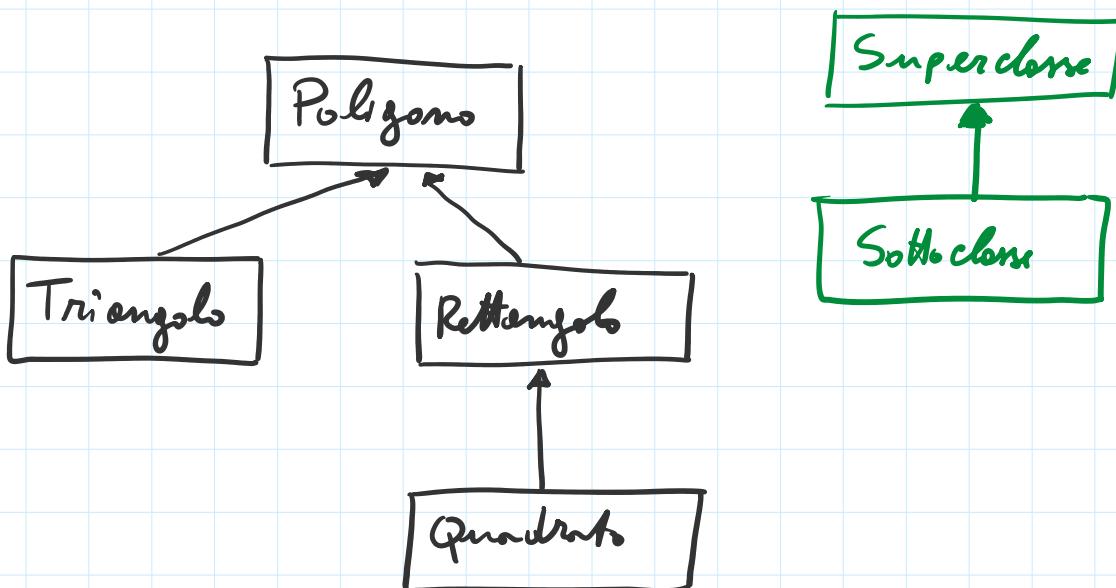
```
class A {  
    =  
}
```

A: classe base, superclasse
classe padre, supertipo

```
class B extends A {  
    =  
}
```

B: classe derivata, sottoclasse
classe figlio, sottotipo

Definiamo B come sole
differenze rispetto ad A



In Java l'ereditarietà è singola:
ogni classe può avere una sola superclasse

ogni classe può avere una sola superclasse

Una classe può avere tante sottoclassi

La gerarchia può avere più livelli

La relazione che esprimiamo è
del tipo

"*è*" *un*"

Un Triangolo è un Poligono

Un Rettangolo è un Poligono

Un Quadrato è un Rettangolo e quindi
anche un Poligono

Un Poligono non è necessariamente
un Triangolo, un Rettangolo o un Quadrato

Quando scrivo che B extends A

- B eredita i metodi e i campi di A

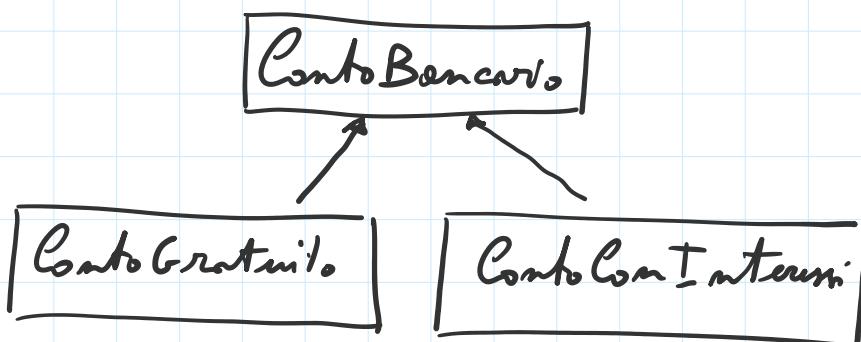
per esprimere le differenze posso

- ridefinire il comportamento dei metodi
definiti in A

- aggiungere nuovi campi e metodi
rispetto ad A

B non può eccedere dei membri
privati di A (onde se vengono ereditati)

Esempio



ContoBancario :

- un intestatario.
- un bilancio
- un identificatore univoco del conto

operazioni possibili:

- depositare
- prelevare
- trasferire
- conoscere le "caratteristiche"
- rappresentazione in formato straniero

```

public class ContoBancario {
    private double bilancio;
    private final String intestatario;
}
  
```

final : una volta impostato non può essere cambiato

```

private final String intestatario;
private final int numero;
private static int prossimo = 1;

public ContoBancario(String i){
    intestatario = i;
    numero = prossimo++;
}

public double getBilancio(){
    return bilancio;
}

public String getIntestatario(){
    return intestatario;
}

public int getNumero(){
    return numero;
}

public void deposita(double d){
    bilancio += d;
}

public void preleva(double d) {
    bilancio -= d;
}

public void trasferisci(ContoBancario c, double d) {
    preleva(d);
    c.deposita(d);
}

public String toString(){
    return "intestatario: " + intestatario +
        ", numero: " + numero +
        ", bilancio: " + bilancio;
}

```

final: il numero del conto non può cambiare

static: è in copia singola per tutti gli oggetti di tipo ContoBancario

*somma e assegnante
equivale a bilancio = bilancio + d*

tipi ContoBancario

Un ContoConInteressi è un ContoBancario caratterizzato dall'aver un tasso di interesse

operazione aggiuntiva:
aggiungere al bilancio gli interessi maturati

```

public class ContoConInteressi extends ContoBancario {
    private final double tasso;
    public ContoConInteressi (String i, double t){
        super(i);
        tasso = t;
    }
    public double getTasso(){
        return tasso;
    }
    public void aggiungiInteressi(){
        double q = getBilancio() * tasso;
        deposita(q);
    }
    public String toString(){
        String s = super.toString();
        return s + ", tasso: " + tasso;
    }
}

```

richiama
il toString
di ContoBancario

← impiù rispetto al
bilancio, numero ...

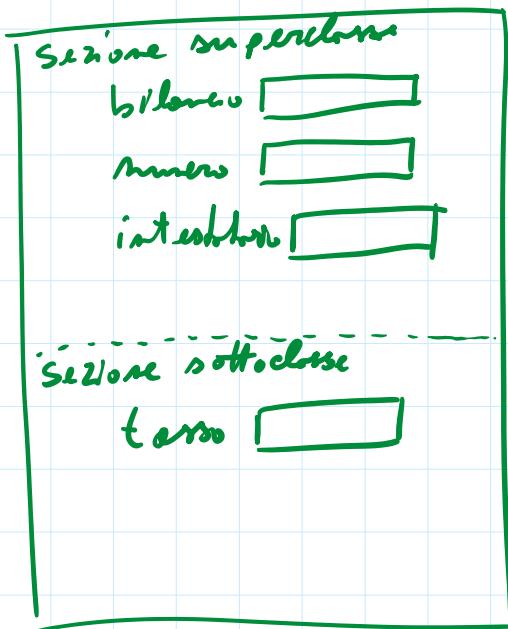
↑ tasso del conto

chiama il
costruttore di
ContoBancario

ereditato da
ContoBancario

ri definisce toString
rispetto alla versione
di ContoBancario
usando il toString
di ContoBancario

ContoConInteressi



Potrete creare un Conto Bancario:

Conto Bancario cb1 = new Conto Bancario ("Mario");

e un Conto Con Interessi

Conto Con Interessi cc1 = new Conto Con Interessi ("Luigi",
0.1);