

**Università di degli Studi della Campania  
Luigi Vanvitelli  
Dipartimento di Ingegneria**

**Programmazione ad Oggetti**

*a.a. 2020-2021*

**Overloading  
Costruttori e Distruttori**

Docente: Ing. Massimo Ficco  
E-mail: [massimo.ficco@unicampania.it](mailto:massimo.ficco@unicampania.it)

1

1

## **Importanza dei nomi**



È importante l'uso dei nomi nello scrivere un programma.  
Quando si crea un oggetto si dà un nome ad una regione della memoria.

Un metodo è un nome associato ad un'azione

Assegnando a metodi e variabili nomi significativi si rende più leggibile il proprio programma



2

# Overload

V:

Un nome è ***Overloaded*** quando assume più significati:

- Lavare la maglia
- Lavare il cane
- Lavare la macchina

Nell'esempio è l'oggetto che dà significato al verbo.



# Elementi di distinzioni

V:

L'overload di un metodo si distingue in base a:

- Il tipo dei parametri
- L'ordine dei parametri

Non è possibile definire un overload in base al valore di ritorno:

- **int f();** o **float f();**

*Si capirebbe quale utilizzare solo se scrivessi sempre  
int x=f();*



# Overload Ex.

V:

```
public class CalculatorTest {  
  
    public static void main(String [] args) {  
        Calculator calc = new Calculator();  
  
        int totalOne = calc.sum(2,3);  
        System.out.println(totalOne);  
  
        float totalTwo = calc.sum(15.9F, 12.8F);  
        System.out.println(totalTwo);  
  
        float totalThree = calc.sum(2, 12.8F);  
        System.out.println(totalThree);  
  
        float totalFour = calc.sum(2L, 12.8F);  
        System.out.println(totalFour);  
    }  
}  
  
public class Calculator {  
  
    public int sum(int one, int two){  
        System.out.println("Method One");  
        return one + two;  
    }  
  
    public float sum(float one, float two) {  
        System.out.println("Method Two");  
        return one + two;  
    }  
  
    public float sum(int one, float two) {  
        System.out.println("Method Three");  
        return one + two;  
    }  
}
```



# Overload Ex.

V:

```
public class ShirtTwo {  
    public void setShirtInfo(int ID, String desc, double cost){  
        shirtID = ID;  
        description = desc;  
        price = cost;  
    }  
  
    public void setShirtInfo(int ID, String desc, double cost, char color) {  
        shirtID = ID;  
        description = desc;  
        price = cost;  
        colorCode = color;  
    }  
  
    public void setShirtInfo(int ID, String desc, double cost, char color, int quantity) {  
        shirtID = ID;  
        description = desc;  
        price = cost;  
        colorCode = color;  
        quantityInStock = quantity;  
    }  
}
```



## Overload Ex.

V:

```
class ShirtTwoTest {  
  
    public static void main (String args[]) {  
        ShirtTwo shirtOne = new ShirtTwo();  
        ShirtTwo shirtTwo = new ShirtTwo();  
        ShirtTwo shirtThree = new ShirtTwo();  
  
        shirtOne.setShirtInfo(100, "Button Down", 12.99);  
        shirtTwo.setShirtInfo(101, "Long Sleeve Oxford", 27.99, 'C');  
        shirtThree.setShirtInfo(102, "Shirt Sleeve T-Shirt", 9.99, 'B', 50);  
  
        shirtOne.display();  
        shirtTwo.display();  
        shirtThree.display();  
    }  
}
```



## Overload .1

V:

```
class Tree {  
    int height=0;  
  
    void info() { System.out.println("Tree is " + height + " feet tall"); }  
  
    void info(String s) { System.out.println(s + ": Tree is " + height + " feet tall"); }  
}  
  
public class Overloading {  
    public static void main(String[] args) {  
        Tree t = new Tree();  
        t.info();  
        t.info("overloaded method");  
    }  
}
```



## Overloading .2

V:

```
public class OverloadingOrder {  
  
    static void print(String s, int i) { System.out.println("String: " + s +  
        ", int: " + i); }  
  
    static void print(int i, String s) { System.out.println("int: " + i + ",  
        String: " + s);}  
  
    public static void main(String[] args) {  
        print("String first", 11);  
        print(99, "Int first");  
    }  
}
```



V:

## Il costruttore



# Il costruttore

V:

La maggior parte degli errori vengono fatti perché ci si dimentica di inizializzare le variabili

Per inizializzare un oggetto:

- Un metodo ad hoc ?
- Il costruttore permette di non dividere creazione ed inizializzazione dell'oggetto



Programmazione ad Oggetti - Prof. Massimo Ficco

11

# Il costruttore in Java

V:

In Java il costruttore:

- Viene creato di default se non definito
- Deve avere lo stesso nome della classe
- Non ritorna niente (neanche void)
- Può avere o non dei parametri
- Viene richiamato da new che restituisce un riferimento all'oggetto chiamato

**Albero a = new Albero()**



Programmazione ad Oggetti - Prof. Massimo Ficco

12

## Il costruttore di default

V:

Se definisco un costruttore senza parametri l'operatore new utilizza quello da me definito

Se non definisco un costruttore senza parametri posso usare quello di default di java che alloca semplicemente l'oggetto



## Esempio 1:

V:

```
public class Albero {  
    int altezza=0;  
    public Albero ()  
    {  
        System.out.println("Costruito un albero di  
        altezza"+altezza);  
    }  
}
```



## Esempio 2

V:

```
public class Albero{
    int altezza=0;

    public Albero (){System.out.println("Costruito un albero di
        altezza"+altezza);}

    public Albero (int h)
    {
        altezza=h;
        System.out.println("Costruito un albero di
            altezza"+altezza);
    }
}
```



Programmazione ad Oggetti - Prof. Massimo Ficco

15

## Overload del Costruttore

V:

```
public class ShirtTest {

    public static void main (String args[] ) {
        Shirt shirtFirst = new Shirt ( );
        Shirt shirtSecond = new Shirt ('G');
        Shirt shirtThird = new Shirt ('B', 1000);
        //...
    }
}

public class Shirt {

    //...
    public Shirt ( ) {
        colorCode = 'R';
    }

    public Shirt (char startingCode) {
        colorCode = startingCode;
    }

    public Shirt (char startingCode, int startingQuantity) {
        colorCode = startingCode;
        quantityInStock = startingQuantity;
    }
    //...
}
```

Programmazione ad Oggetti - Prof. Massimo Ficco

16

16



V:

## Il distruttore



## Il distruttore

V:

Gli oggetti vengono creati dinamicamente

Finito l'ambito di visibilità lo spazio di memoria associato deve essere deallocato

Nel caso di JAVA si occupa di questo il “**garbage collector**”

- Un oggetto può essere deallocato quando nessun riferimento punta più a quell'oggetto
- Non è determinabile a priori il momento in cui interverrà il *garbage collector*



## Il distruttore Esempio.1

V:

```
String a=new String("casa");  
String b=a;  
a=null; //b punta ancora all'oggetto  
b=null; //oggetto può essere distrutto
```



Programmazione ad Oggetti - Prof. Massimo Ficco

19

## Operazioni di finalizzazione V:

Se serve effettuare delle operazioni prima che l'oggetto venga distrutto è possibile definire un metodo **finalize**:

- Esso viene richiamato appena prima l'operazione di deallocazione



Programmazione ad Oggetti - Prof. Massimo Ficco

20

## Distruttore esempio .2

V:

```
class Book {
    boolean checkedOut = false;
    Book(boolean checkOut) { checkedOut = checkOut; }
    void checkIn() { checkedOut = false; }
    public void finalize() { if(checkedOut) System.out.println("Error: checked out");}
}

public class TerminationCondition {
    public static void main(String[] args) {
        Book novel = new Book(true);
        // Proper cleanup:
        novel.checkIn();
        // Drop the reference, forget to clean up:
        novel = new Book(true);
        // Force garbage collection & finalization:
        System.gc();
    }
} //:~
```



Programmazione ad Oggetti - Prof. Massimo Ficco

21

## Un esempio significativo !

V:

```
public class Oggetto{
    static int n_obj=0;
    public Oggetto() { n_obj++; }
    public void finalize() { n_obj--; }
}
```

La variabile **n\_obj** conta il numero di istanze allocate per la classe oggetto !!!



Programmazione ad Oggetti - Prof. Massimo Ficco

22

V:

---

## Persistenza e visibilità



## Persistenza e visibilità

---

V:

Una variabile di un metodo è valida fino a quando non termina il metodo.

Un attributo di un oggetto è valido fino a quando è vivo l'oggetto

- L'oggetto viene creato
- **L'oggetto viene distrutto** e la memoria che impegna deallocata quando nessun riferimento punta più a quell'oggetto

