UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Università di degli Studi di Salerno
Dipartimento di Informatica

Programmazione ad Oggetti

a.a. 2023-2024

Classi e Oggetti

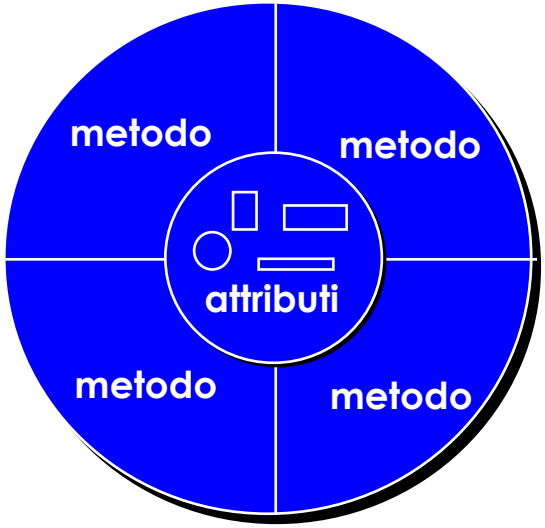
Docente: Massimo Ficco

E-mail: *mficco@unisa.it*

Tipo Dati Astratto

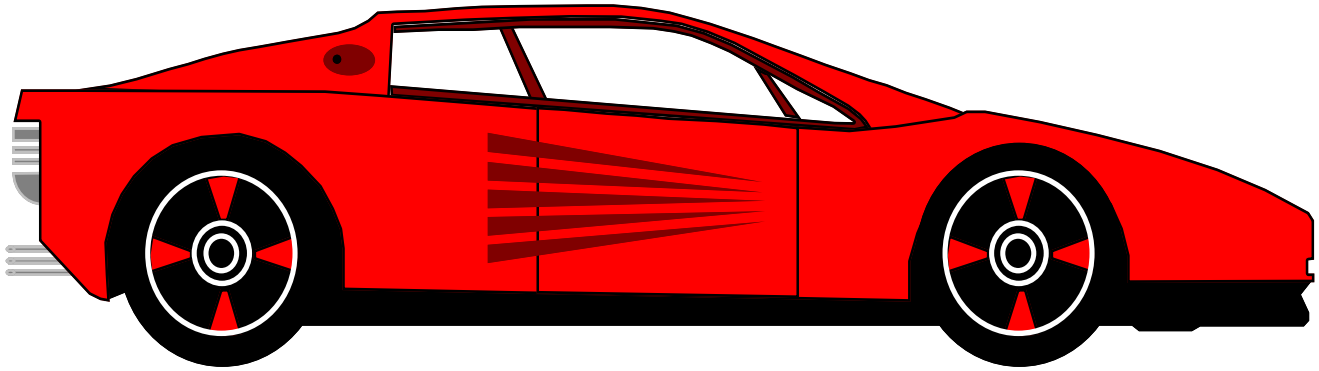
ASTRAZIONE SUI DATI

Il tipo di dati astratto (ADT)



- **una classe ha un <u>nome</u>, e contiene due tipi di membri: <u>attributi</u> e <u>metodi</u>**

ESEMPIO: UN'AUTOMOBILE



Funzioni

- Avviati
- Fermati
- Accelera
- …

Dati:

- Targa
- Colore
- Velocità
- Livello benzina
- …

Le classi

Java è un linguaggio di ***programmazione orientato agli oggetti*** (*object oriented*), che si basa sul concetto di classe

La *classe* è identificata univocamente da un nome, che deve essere necessariamente uguale al nome del file che la contiene (affinché la classe possa essere eseguita)

La classe ed è caratterizzata da una coppia di parentesi graffe **{ }** che contiene il corpo della classe ossia le istruzioni e le dichiarazioni

Esempio: (File sorgente con nome ***Prova.java***)

```
[modifiers] class  Prova {
        corpo della classe
}
```

I nomi delle classi possono essere precedute da parole chiave [modifiers]: **public**, **private**, **static**, …

Le classi in JAVA

Definizione di una classe vuota:

```
public class Auto {
        /* Class body goes here */
}
```

Creazione di un oggetto:

```
Auto panda = new Auto();
```

Non posso fare niente con una classe vuota!!!!!

Occorre aggiungere attributi e metodi !!!!

GLI ATTRIBUTI

[modificatore] Tipo nomeAttributo = [Valore]

Il ***modificatore*** può essere definito: **public, privato,** …..
(per default è public)

Esempio classe auto

```
Public class Auto {
        public int  cilindri=4;
        public int  speed;
        public String targa;
}

Auto panda=new Auto();
```

Ho una classe costituita solo di attributi !!!!!!

Unica cosa che possiamo fare è accedere agli attributi per modificarli o utilizzarli:

```
panda.speed =10;

System.out.println ( panda.cilindri );
```

I metodi

Ogni classe può contenere uno o più metodi

I metodi sono caratterizzati da un nome e una coppia di graffe { } che contiene la specifica procedura

Syntax

```
[modifiers]return_type method_identifier([arguments])
{ method_code_block }
```

I nomi dei metodi possono essere precedute da parole chiave [modifiers]: ***public***, ***private***, ***static,*** ..

```
public class Auto {
        public int  cilindri=4;
        public int  speed=0;
        public String targa;
        public void accellera () { speed++; }
}
```

I parametri di ritorno

```
// questo metodo ritorna il numero di byte della stringa s
public int storage (int s) {
        return  s = 2;
}
```

Oggetti e tipi semplici vengono passati allo stesso modo
Il parametro si utilizza nel metodo come una qualunque variabile
return ha due funzioni:

- Terminazione
- Restituisce il valore di ritorno

Esempi di terminazione

```
public boolean flag() { return true; }

public float naturalLogBase() { return 2.718f; }

public void nothing() { return; }

public void nothing2() {}
```

Esempio classe auto

```
class Auto{
        public int  cilindri=4; //Inizializzata esplicitamente
        public int  speed=0;  //Inizializzata esplicitamente
        public String targa;   //Inizializzata automaticamete???

        public int getSpeed(){return speed;};
        public void setSpeed(int s){speed= s;};
        public int getCilindri(){return cilindri};
        // Due alternative ???
        public void setTarga(){targa = "XF345PF"; }
        public void setTarga(String s){ targa = s;}
}
```

Chiamata di un metodo

Esempio:

- Supponiamo di avere un metodo f**( )** che non ha parametri e ritorna in tipo **int**. → **public int f()**
- Supponendo di aver un oggetto **a** per il quale è possibile chiamare **f( )** → **int x = a.f();**

panda.setSpeed(100);

int s = panda.getSpeed();

Il tipo ritornato da f deve essere compatibile con il tipo di x

VARIABILI LOCALI DI UN METODO

VARIABILI LOCALI DI UN METODO

- Le variabili locali a un metodo:
  - sono visibili solo dal corpo del metodo
  - vengono allocate (nello stack di run-time) alla chiamata e deallocate all'uscita del metodo
  - non vengono inizializzate automaticamente (diversamente dai campi di una classe)

- Non si può accedere a una variabile a cui non si sia prima assegnato un valore (e viene segnalato in compilazione !)

Esempio:

```
int  i;
if ( cond ) { i = 55; … }
i++;     /* compile-time error */
```

VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

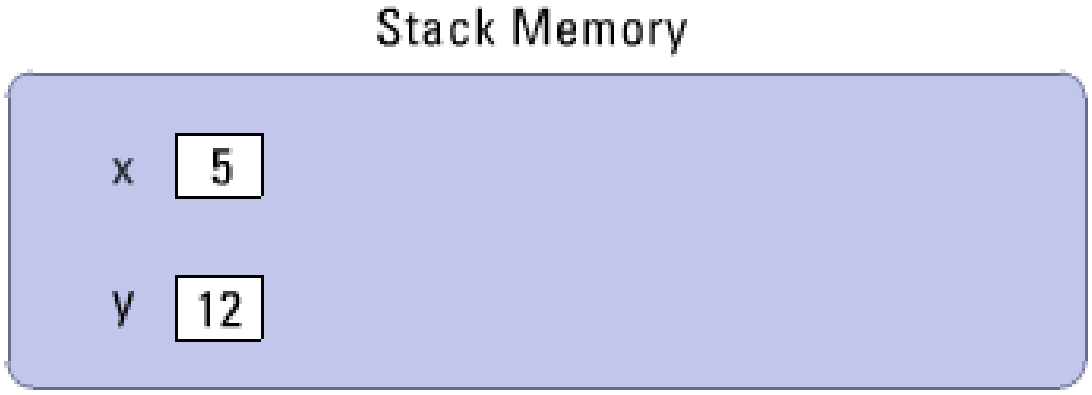
foo()

Stack Memory

VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

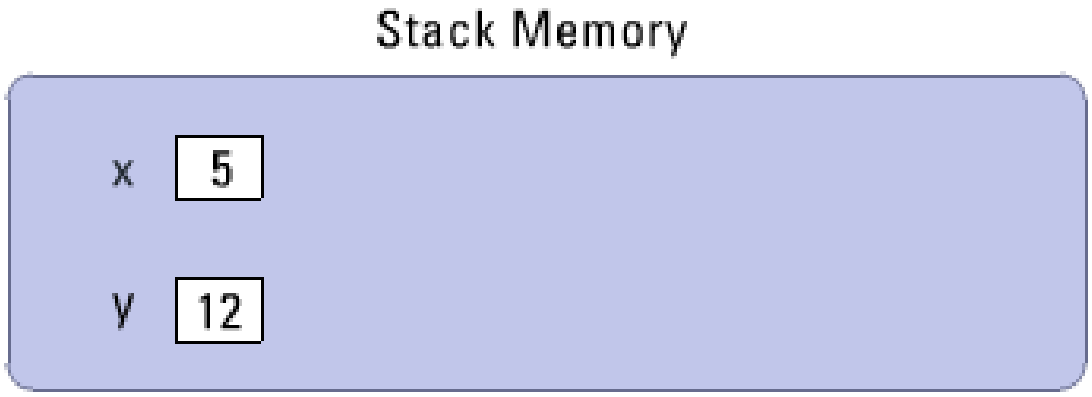
foo()



VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

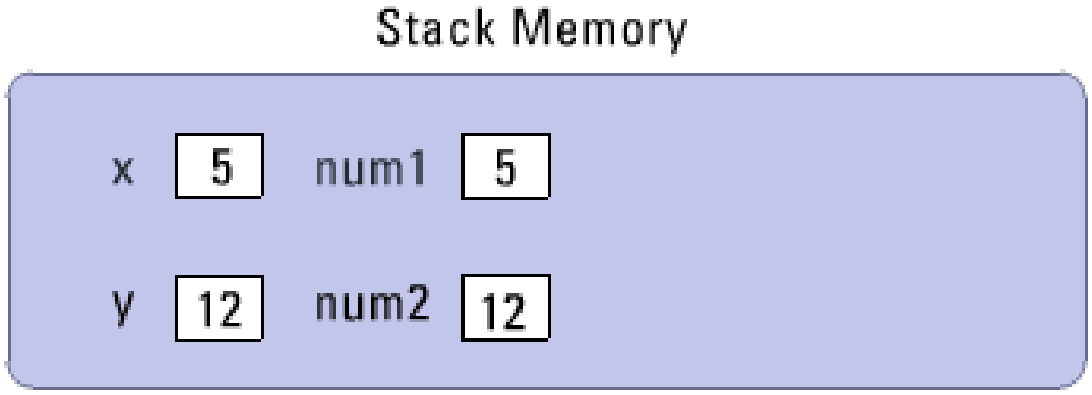
foo()



VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

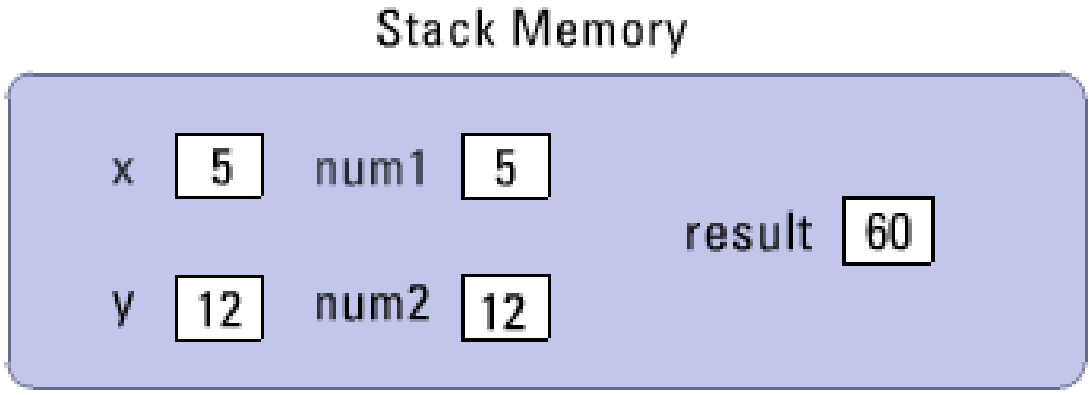
foo()



VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

foo()



VARIABILI LOCALI DI UN METODO

```
public void multiply (int num1, int num2) {
   int result = num1* num2;
}
public void foo ( ) {
   int x = 5, y = 12;
   multiply (x, y);
}
```

foo()

Stack Memory

SCAMBIO DI PARAMETRI

Tutto viene passato per valore

La lista dei parametri deve specificare il tipo e l'ordine

Il compilatore da errore se il tipo del parametro passato è diverso da quello definito nel prototipo.

Dei tipi semplici viene passato il valore - (int, boolean, char, ..)

Degli oggetti viene passato il riferimento (praticamente il puntatore) - (array, stringhe, oggetti di utente…).

Come si fa a passare un parametro di uscita ?????

Solo oggetti possono essere parametri di uscita

Perché ? In che modo ?

Varargs

- Dalla versione 1.5 in Java sono disponibili i **varargs**, un metodo per avere metodi con zero o più parametri di uno specifico tipo.

```
public static int somma(int... Args) {
        int sum = 0;
        for (int arg : Args)
        {
            sum += arg;
        }
        return sum;
    }
```

- Internamente il tutto viene gestito creando un array la cui dimensione è il numero di argomenti passati e il cui contenuto sono gli argomenti stessi, array che poi è passato al metodo.
- Se si utilizzano i varargs, ad ogni invocazione del metodo in questione abbiamo l'allocazione e l'inizializzazione di un array. Se è possibile stimare statisticamente il numero di parametri è possibile impiegare l'overloading del metodo.

L'esempio più semplice
HELLO WORLD

Affinché un applicazione ad oggetti "parta", devo avere una classe con un metodo statico e pubblico di nome **main**

Hello.java

```
class Hello {
   public static void main (String args [ ])  {
         System.out.println("Hello World!");
   }
}
```

obbligatorio in questa forma

SPIEGAZIONI

public static void main (String args [])

- ***void*** indica che *main* non ritorna nulla, il che è necessario per superare il type-checking del compilatore
- ***args[]*** sono gli argomenti passati a *main* dalla shell quando si digita: ***java Hello arg1 arg2 ... argn***
- *String* dice che gli argomenti sono di classe ***String***
- ***public*** rende il metodo main visibile alle altre classi - e al comando *java* (interprete)
- ***static*** associa *main* alla classe *Hello*, e non alle sue istanze

System.out.println("HelloWorld!")

- invoca il metodo *println* dell'oggetto *out* della classe *System*, che stampa la stringa sul file *stdout*

Esempio preliminare

```
class Contatore {
        int cont=0;

        public void incr() {  cont++; }

        public static void main(String args[]) {
                Contatore a=new Contatore();
                Contatore b=new Contatore();

                a.incr();
                System.out.println(a.cont);                output ?
                System.out.println(b.cont);                output ?
                b=a; // copia i riferimenti
                System.out.println(a.cont);                output ?
                System.out.println(b.cont);                output ?
                b.incr();
                System.out.println(a.cont);                output ?
                System.out.println(b.cont);                output ?
        }
}
```