Corso di Programmazione a oggetti

Introduzione alla programmazione a oggetti

a.a. 2014/2015
Francesco Fontanella

La Programmazione Modulare

- Un programma può essere visto come un insieme di moduli che interagiscono tra di loro. Tutte le funzioni di un modulo sono scritte nello stesso file.

- Ogni modulo è fatto di due parti, la specifica e l'implementazione. Questi due aspetti sono separati mettendoli in due file diversi: la specifica si trova nel suo file .h mentre l'implementazione nel file .cpp.

MODULO


interfaccia
(Visibile dall'esterno)
corpo
(Nascosto all'esterno e protetto)

- I moduli non fanno parte del linguaggio, ma sono un'organizzazione del codice da parte del programmatore.

- La programmazione modulare offre enormi vantaggi nell'implementazione di programmi, in particolare al crescere della complessità dei programmi da scrivere.

Programmazione Modulare: vantaggi

- Suddivisione del lavoro tra diversi programmatori;
- Semplificazione del riuso del software
- Semplificazione della manutenzione;

La Programmazione ad Oggetti

- La programmazione ad oggetti rappresenta un ulteriore sviluppo rispetto alla programmazione modulare.

- La programmazione orientata agli oggetti (Object Oriented Programming, OOP) è un paradigma di programmazione, in cui un programma viene visto come un insieme di oggetti che interagiscono tra di loro.

Un paradigma di programmazione è caratterizzato dall'insieme di concetti e astrazioni che utilizza per rappresentare un programma e i passi che compongono un calcolo.

Esempi di paradigmi sono: imperativo, funzionale, object oriented, logico:

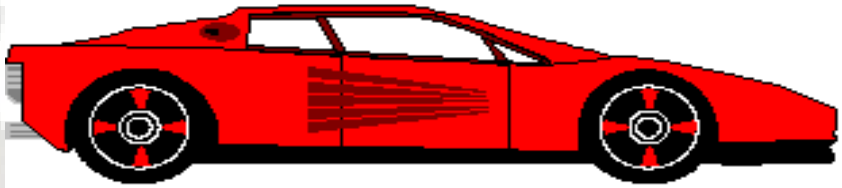
- Nei linguaggi OOP esiste un nuovo tipo di dato, la *classe*. Questo tipo di dato serve appunto a modellare un insieme di oggetti dello stesso tipo.

- In generale, un oggetto è caratterizzato da un insieme di attributi e da un insieme di funzionalità

Un Esempio di Oggetto

- Un automobile può essere caratterizzata in questo modo:

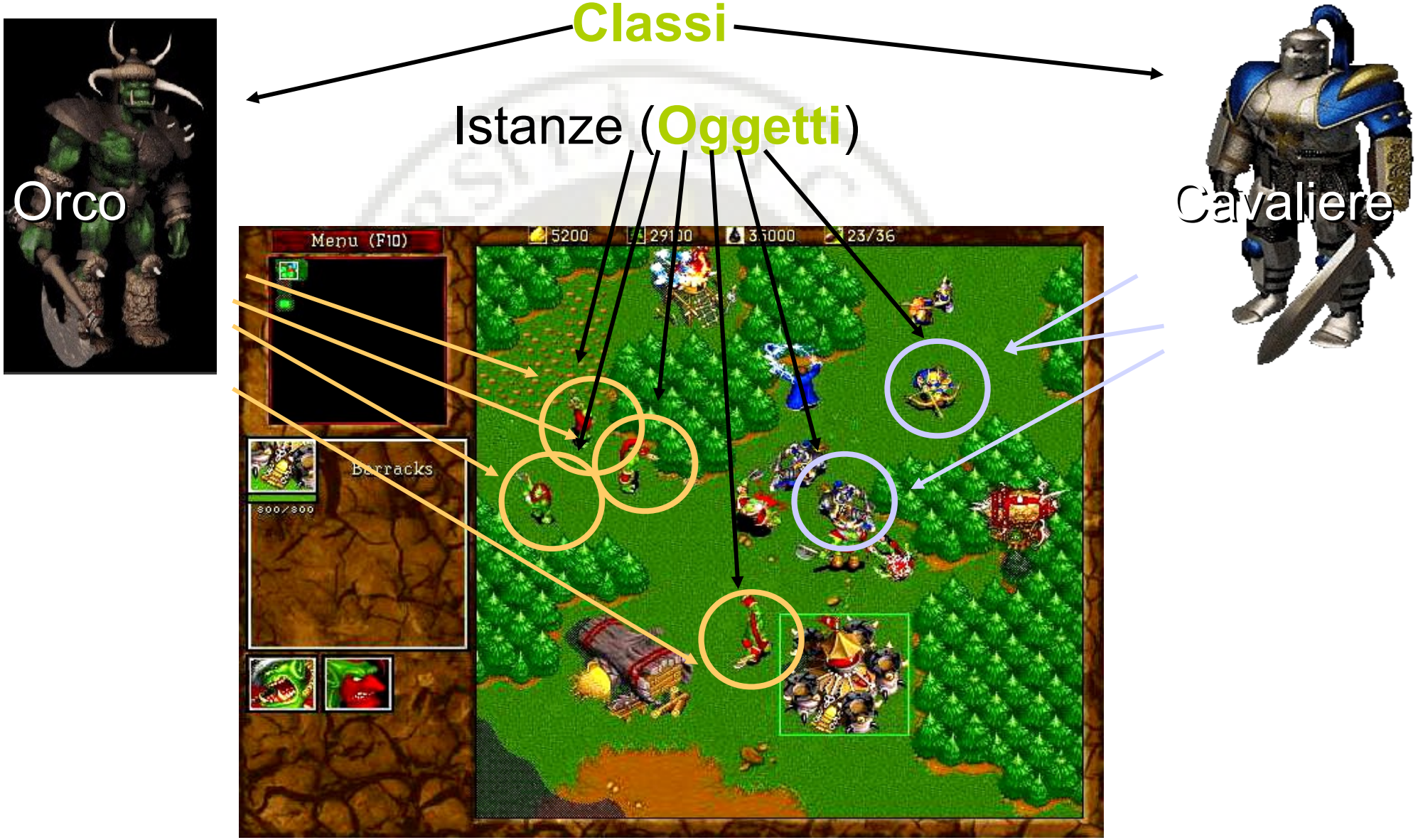
attributi	**funzionalità**
potenza	frena
marce	accellera
peso	cambio marcia
cilindrata	cambia direzione



- È possibile interagire con l'automobile, per determinarne il suo comportamento attraverso il suo interfaccia che permette di effettuare le operazioni consentite:
  - Pedale del freno
  - Pedale dell'acceleratore
  - Leva del cambio
  - volante

Un Ulteriore Esempio: un Videogioco

Classi e Oggetti: un Esempio



La Programmazione Orientata agli Oggetti

- La programmazione orientata agli oggetti si basa su alcuni concetti fondamentali:
  - **Classe**
  - **Incapsulamento**
  - **Oggetto**
  - **Ereditarietà**
  - **Polimorfismo**

Linguaggi ad Oggetti

- Un linguaggio di programmazione ad oggetti offre **costrutti espliciti** per la definizione di entità (oggetti) che incapsulano una struttura dati e le operazioni possibili su di essa.

- Alcuni linguaggi, in particolare il C++, consentono di definire tipi astratti, e quindi istanze (cioè, variabili) di un dato tipo astratto. In tal caso il linguaggio basato sugli oggetti presenta costrutti per la definizione di classi e di oggetti.

Classi e oggetti

- Nei linguaggi a oggetti, il costrutto *class* consente di definire nuovi tipi di dato e le relative operazioni.

- Le operazioni possono essere definite sotto forma di operatori o di funzioni (dette metodi o funzioni membro), i nuovi tipi di dato possono essere gestiti quasi allo stesso modo dei tipi predefiniti del linguaggio:
  - si possono creare istanze
  - si possono eseguire operazioni su di esse

- Un oggetto è una variabile di un particolare tipo di dato definito dall'utente per mezzo del costrutto class.

- Nel gergo dei linguaggi OO una variabile definita di un certo tipo (classe) rappresenta **un'istanza** della classe.

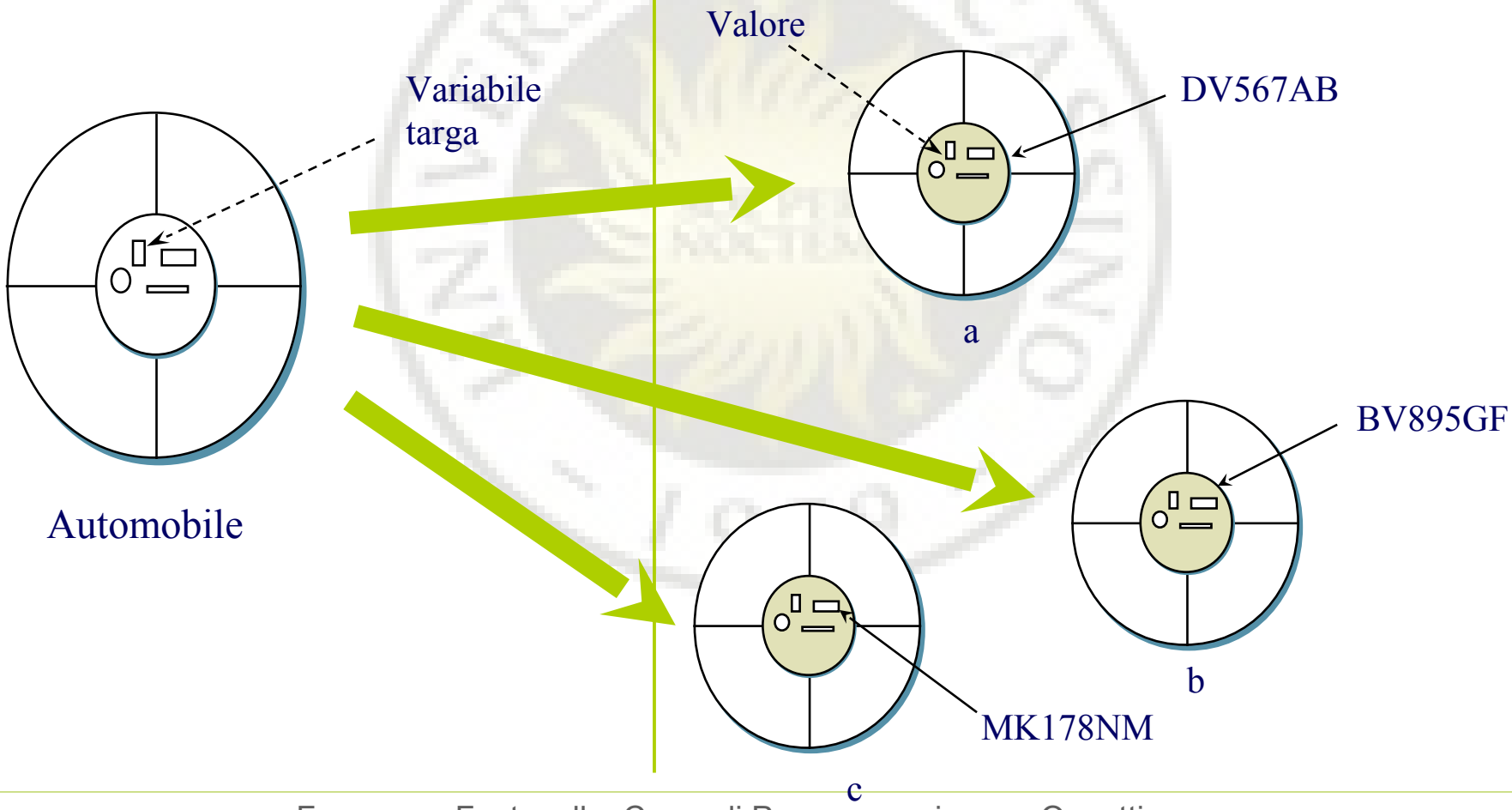
- Lo **stato** di un oggetto, invece, è rappresentato dai valori correnti delle variabili che costituiscono la struttura dati utilizzata per implementare il tipo di dato rappresentato definito dalla classe.

L’istanziazione degli Oggetti

- Un oggetto è una istanza (“esemplare”) di una classe.
- Due esemplari della stessa classe sono distinguibili soltanto per il loro stato (i valori dei dati membro), mentre il comportamento (definto dalle funzioni membro) è sempre identico.

CLASSE

ISTANZE



Caratteristiche di una Classe

- La classe è un modulo software con le seguenti caratteristiche:
  - E' dotata di un'interfaccia (specifica) e di un corpo (implementazione);
  - La struttura dati "concreta" di un oggetto della classe, e gli algoritmi che ne realizzano le operazioni, sono tenuti nascosti all'interno del modulo che implementa la classe;

- Lo stato di un oggetto evolve unicamente in relazione alle operazioni, definite nell'interfaccia, ad esso applicate
- Le operazioni sono utilizzabili con modalità che prescindono completamente dagli aspetti implementativi
- in tal modo è possibile modificare gli algoritmi utilizzati senza modificare l’interfaccia

Il Linguaggio C++

- C++ è un linguaggio di programmazione general-purpose che supporta più paradigmi di programmazione :
  - la programmazione procedurale (è un C “evoluto”);
  - la programmazione orientata agli oggetti;
  - la programmazione generica.
- Per questo motivo il C++ è quindi un linguaggio ibrido

L'Incapsulamento

- L'incapsulamento (in inglese: *information hiding*) è la suddivisione di un oggetto in due parti:
  - **Interfaccia:**
    - costituito dall'insieme di **metodi** (detti anche messaggi) che possono essere invocati dall'utente per accedere alle funzionalità dell'oggetto;
  - **Implementazione**
    - Contiene l'implementazione delle funzionalità dell'oggetto e delle strutture dati necessarie
    - È nascosta all'utente

- L'incapsulamento è realizzato per mezzo delle istruzioni:
  - **Private:** le funzioni e le variabili definite private NON sono accessibili all'esterno della classe.
  - **Public:** le funzioni e le variabili definite publiche sono accessibili all'esterno della classe.

Le classi in C++

- Il linguaggio C++ supporta esplicitamente la dichiarazione e la definizione di tipi astratti da parte dell'utente mediante il costrutto *class*; le istanze di una classe vengono dette oggetti.

- In una dichiarazione class occorre specificare sia la struttura dati che le operazioni consentite su di essa. Una classe possiede, in generale, una sezione pubblica ed una privata.

- La sezione pubblica contiene tipicamente le operazioni (dette anche metodi) consentite ad un utilizzatore della classe. Esse sono tutte e sole le operazioni che un utente può eseguire, in maniera esplicita od implicita, sugli oggetti.

- La sezione privata comprende le strutture dati e le operazioni che si vogliono rendere inaccessibili dall'esterno.

Le classi in C++: un Esempio

Inter faccia

Implem entazione

```
class Contatore {
  public:
    void Incrementa();
    void Decrementa();
    unsigned int get_value();

  private:
    unsigned int value;     // valore corrente
    const unsigned int max; // valore massimo
};
```

La Specifica di una Classe

- Rappresenta un'interfaccia per la classe
- in essa vengono dichiarati sia i nomi dei metodi (funzioni) della classe che i tipi delle variabili della classe, sia pubblici che privati
- Permette l'uso della classe senza che l'utente conosca i dettagli di implementazione (programmazione modulare).
- E' scritta in un apposito "file di intestazione".

Specifica: Notazione Base

```
//nome del file C.h
class C {
 public:
   //prototipi delle funzioni membro
   T1 f1(....);
   T2 f2(....);
        .........
   private:
  //struttura dati
    int i;
    char c;
          ........
}; //fine specifica della classe C
```

- Di **default,** i.e. se non specificato diversamente, tutti gli attributi di una classe sono **privati** (per le struct è il contrario)
- Pertanto la classe dell' esempio precedente può essere scritta così:

```
class C {
   //strutture dati
    int i;
    char c;
    .....
 public:
  T1 f1(....);
   T2 f2(....);
              .....

};
```

L’Implementazione di una Classe

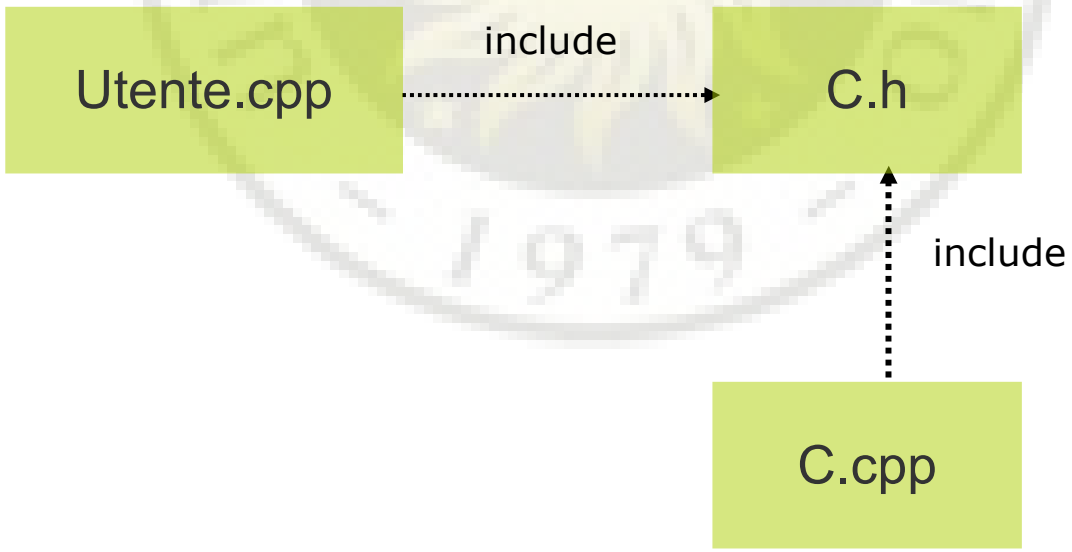
- E' la codifica in C++ dei metodi della Classe.
- E' una particolare soluzione (può cambiare l'implementazione senza che cambi l'interfaccia)
- E' scritta in un apposito "file di implementazione " (solitamente con estensione .cpp oppure .cc)

```
//nome del file C.cpp
#include "C.h"
T1  C::f1(....) {
   // implementazione della funzione f1
  ...
}
T2  C::f2(....) {
// implementazione della funzione f2
  ...
}
//fine del file C.cpp
```

deve includere l'header file

Uso di una Classe

- Affinché un programma possa utilizzare un classe deve:
  - includere la specifica della classe (contenuta nel file nomeClasse.h)
  - dichiarare istanze della classe.



```
#include "C.h"

main() {
  C c1, c2; // definizione degli oggetti c1 e c2          // della
   classe C
  ...
  c1.f1(....); // si applica la funzione membro f1
                 // all'oggetto c1
  c1.f1(....); // si applica la funzione membro f2
                 // all'oggetto c1
  c2.f1(....); // si applica la funzione membro f1
                 // all'oggetto c2
    .
    .
    .
} //fine programma utente
```

Vantaggi della programmazione Object Oriented

- **modularità**: le classi sono i moduli del sistema software;
- **coesione dei moduli**: una classe è un componente software ben coeso in quanto rappresentazione di una unica entità;
- **disaccoppiamento dei moduli**: gli oggetti hanno un alto grado di disaccoppiamento in quanto i metodi operano sulla struttura dati interna ad un oggetto; il sistema complessivo viene costruito componendo operazioni sugli oggetti;

- **information hiding**: sia le strutture dati che gli algoritmi possono essere nascosti alla visibilità dall'esterno di un oggetto;
- **riuso**: l'ereditarietà consente di riusare la definizione di una classe nel definire nuove (sotto)classi; inoltre è possibile costruire librerie di classi raggruppate per tipologia di applicazioni;
- **estensibilità**: il polimorfismo agevola l'aggiunta di nuove funzionalità, minimizzando le modifiche necessarie al sistema esistente quando si vuole estenderlo.

I Costruttori

- È molto comune che una parte dell'oggetto debba essere inizializzata prima dell'uso. Per esempio nel caso della classe *Contatore* vista in precedenza è necessario inizilizzare il valore della variabile *value.*

- Poiché l'inizializzazione degli oggetti è un'operazione molto comune, il C++ consente di inizializzare gli oggetti al momento della loro creazione.

- Questa inizializzazione automatica è possibile utilizzando le funzioni **costruttore.**

- Un costruttore è una particolare funzione membro di una classe che porta lo stesso nome della classe.

Costruttori: Esempio

```
class Contatore {
  public:
    Contatore();
          .
          .
};
```

```
Contatore::Contatore()
{
 value = 0;
}
```

NOTA

Per le funzioni costruttore non deve essere specificato il tipo restituito, in quanto in C++ le funzioni costruttore non possono restituire valore

- la dichiarazione di una variabile di tipo oggetto, non è solo un'istruzione per così dire passiva di allocazione di memoria, ma implica l'esecuzione del codice contenuto nel costruttore della classe.

- **Esempio**

```
#include “Contatore.h”
Contatore cont1, cont2;
```

Il costruttore viene eseguito per ognuna delle variabili dichiarate

Allocazione dinamica

- I costruttori vengono chiamati anche per gli oggetti allocati dinamicamente.
- In questo caso, il costruttore viene chiamato al momento dell'allocazione.
- **Esempio**

```
#include “Contatore.h”
Contatore *cont_ptr;

cont_ptr = new Contatore;
```

Il costruttore NON viene eseguito quando si dichiarano variabili di tipo puntaore

Il costruttore viene eseguito al momento dell'allocazione dinamica

Costruttori parametrizzati

- I costruttori possono ricevere anche degli argomenti.
- Normalmente lo scopo degli argomenti è quello di passare un valore di inizializzazione.
- **Esempio**

```
class Contatore {
  public:
    Contatore();
    Contatore(int val);
          .
          .
};
```

```
Contatore::Contatore()
{
 value = 0;
}

Contatore::Contatore(int val)
{
 value = val;
}
```

- È possibile definire costruttori con più parametri.

- **Esempio**

```
Class Myclass {
 public:
   Myclass(int i, int j);

          .
          .
 private:
   int a;
    int b;
};
```

```
Myclass::Myclass(int i, int j)
{
 a = i;
 b = j;
}
```

- I costruttori parametrizzati possono essere utilizzati in fase di dichiarazione, o allocazione utilizzando la seguente sintassi:

```
#include “Contatore.h”
#include “Myclass.h”

Contatore cont1, cont2(100), *cont_ptr;
Myclass  mc(0,0), *m_ptr;

cont_ptr = new Contatore(10);
m_ptr = new Myclass(1, 10);
```

Costruttori con un solo parametro

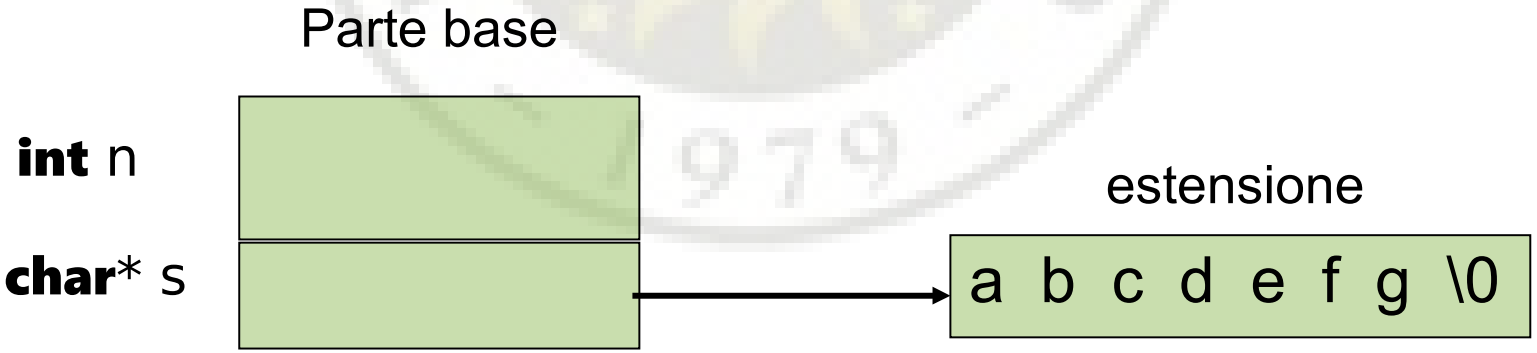
- Nel caso dei costruttori con un solo parametro è consentita anche la seguente sintassi:

```
#include “Contatore.h”
Contatore cont = 100;
```

Viene chiamato il costruttore con parametro uguale a 100

Struttura degli Oggetti

- Ciascun oggetto della classe è costituito:
  - da una parte base, allocata per effetto della definizione dell'oggetto nel programma utente in area dati statici, stack o heap, in base alla classe di memorizzazione;
  - da una eventuale estensione, allocata nell' area heap



Distruttori

- Un distruttore è una funzione membro che:
  - è necessaria solo se l'oggetto presenta un'estensione dinamica
  - ha lo stesso nome della classe, preceduto da ~ (tilde)
  - non restituisce risultato (neanche void)
  - non ha alcun parametro

- Scopo dei distruttori è quello di deallocare l’estensione dinamica di un oggetto

- NON può essere invocata esplicitamente dal programma utente, ma viene invocata implicitamente dal compilatore quando viene deallocato lo spazio di memoria assegnato all'oggetto.

Distruttori: esempio

stack.h

```
Class Stack {
  public:
    Stack();
    ~Stack();
          .
        .

  private:
    int *st_ptr;
    int num;
          .
          .
};
```

stack.cpp

```
// Costruttore
Stack::Stack()
{
 st_ptr = new int[SIZE];
 num = 0;
}

// Distruttore
Stack::~Stack()
{
 delete [] st_ptr;
}
      .
      .
```

Allocazione dinamica del vettore

Dealloca il vettore dinamico

Esecuzione dei Distruttori

- I distruttori vengono eseguiti anche quando vengono deallocati oggetti precedentemente allocati dinamicamente. Es:

```
void funz()
{
  Stack *st_ptr;
      .
      .
  st_ptr = new Stack;
      .
      .
  delete st_ptr;
}
```

Viene invocato il distruttore di stack

Costruttori per variabili locali

- La funzione costruttore di un oggetto locale (definito all'interno di una funzione o anche di un blocco di istruzioni) viene eseguita nel punto in cui si trova la dichiarazione dell'oggetto stesso.

- In pratica i costruttori vengono chiamati subito dopo che è stato allocato spazio sullo stack per l'oggetto.

Distruttori per variabili locali

- Le funzioni distruttori per gli oggetti locali vengono eseguite in ordine inverso rispetto alle funzioni costruttore.
- In pratica esse vengono eseguite subito prima di deallocare lo spazio sullo stack che ospita la variabile.

```
#include Stack.h

void funz()
{
  Stack s1, s2;
        .
        .
        .

  return;
 }
```

Il costruttore viene chiamato prima su s1 e poi su s2

Il distruttore viene chiamato prima su s2 e poi su s1

Costruttori e distruttori: variabili globali

- Le funzioni costruttore degli oggetti globali vengono eseguite **prima** che inizi l'esecuzione del main.

- I costruttori globali vengono chiamati nell'ordine di dichiarazione nel file.

- Non è possibile conoscere l'ordine di esecuzione dei costruttori di oggetti globali specificati in file diversi.

- I distruttori globali vengono eseguiti in ordine inverso **dopo** il termine dell'esecuzione del main.

```
#include Stack.h


  .
  .
Stack s1;

main()
{
    .
    .
    .

  return;
}
```

Il costruttore su s1 viene chiamato PRIMA dell'esecuzione del main!

Il distruttore su s1 viene chiamato DOPO la conclusione del main!