



UNIVERSITÀ
degli STUDI
di CATANIA

Aggregati e collezioni di oggetti

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella

Web: <http://www.dmi.unict.it/farinella>

Email: gfarinella@dmi.unict.it

Dipartimento di Matematica e Informatica

Relazioni “part-of”

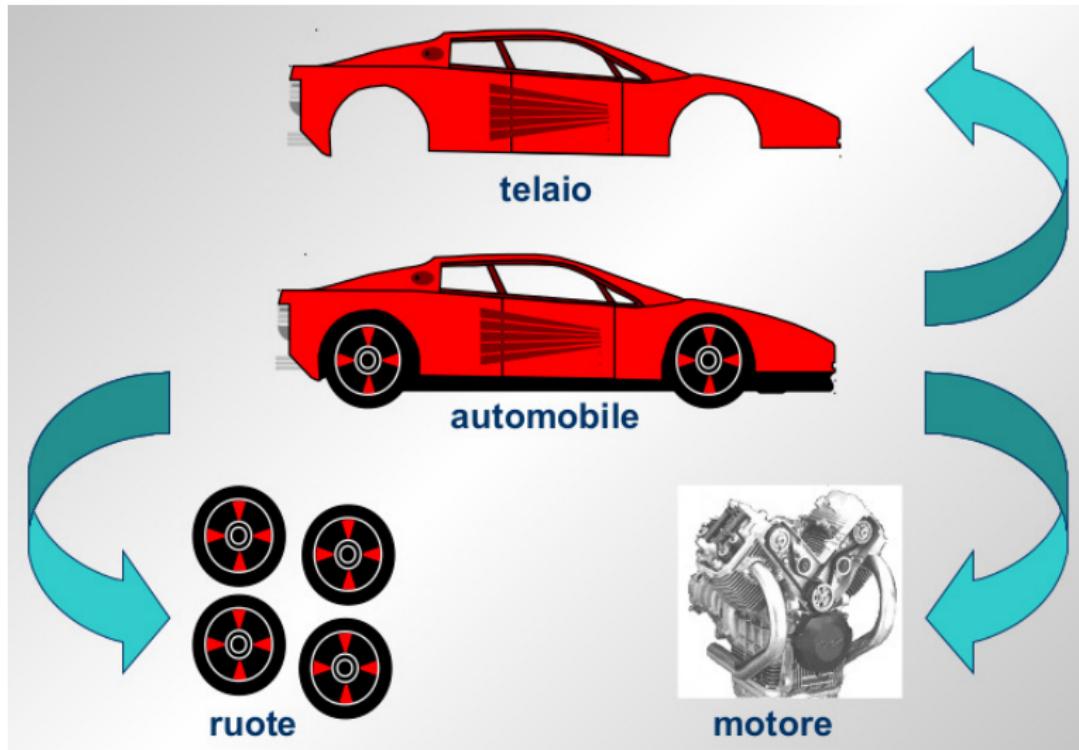
Se una entità da modellare ad oggetti è **complessa**, è conveniente scomporla in un insieme di costituenti:

1. codifica: lo sviluppatore **concepirà diverse classi** per la modellazione della singola entità;
2. esecuzione del programma: un **oggetto sarà composto da più oggetti**, istanze delle classi che insieme modellano l'entità da rappresentare.

Molteplici vantaggi, tra cui:

- **Riuso del codice.** ES: Classe automobile vs classe motore: la classe motore si potrà usare per una differente classe automobile
- **Maggiore manutenibilità:** apportare modifiche al software;

Relazioni “part-of”



Relazioni “part-of”

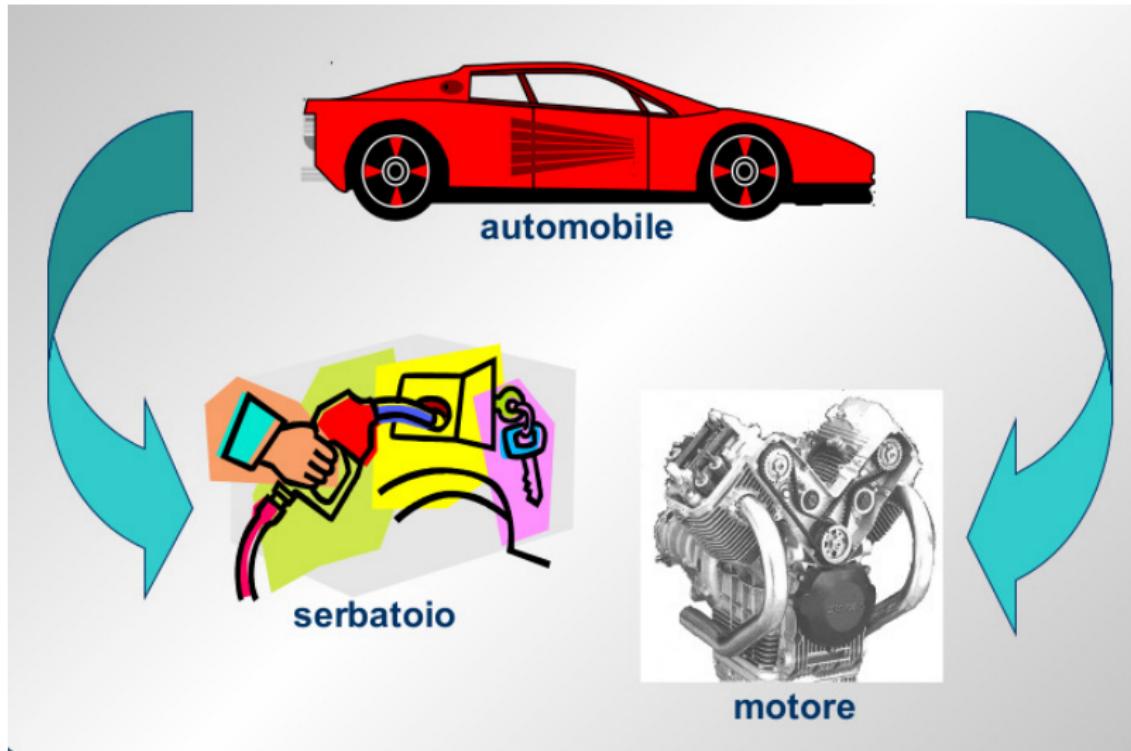
L'oggetto composto scambia messaggi con le sue componenti (gli oggetti che lo compongono):

- il **sistema di avviamento** dell'automobile **comanda**, tra le altre cose, **l'accensione del motore**;
- Il **galleggiante del serbatoio** del carburante invia **messaggi informativi** all'indicatore di livello nel cruscotto;

Relazioni “part-of”

- oppure l'indicatore di livello nel cruscotto invia **messaggi interrogativi** al galleggiante nel serbatoio del carburante per conoscerne lo stato;

Relazioni “part-of”



Relazioni “part-of”

Motore e Serbatoio sono alcune delle componenti della classe Automobile.

Automobile

```
class Automobile {  
    // ...  
    Serbatoio serbatoioCarburante;  
    Motore motoreABenzina;  
}
```

...nella dichiarazione della classe Automobile lo sviluppatore inserirà opportuni campi/attributi che rappresentano istanze delle suddette classi.

Relazioni “part-of”

Interazioni: Automobile → motore e automobile → serbatoio

```
1  bool Automobile::percorri (int km){    Km  
2      //carburante necessario          10     100  
3      float carb =  
4          motoreABenzina.consumoCarburantePerKm(km);  
5  
6      if (carb <= serbatoioCarburante.getQuantita()){  
7          serbatoioCarburante.preleva(carb);  
8          contaKm += km; // stato automobile  
9          cout << "Percorsi km " << km << endl;  
10         return true;  
11     }  
12     else  
13         return false;  
14 }
```

Relazioni “part-of”

```
class Automobile {  
    // ...  
    Serbatoio serbatoioCarburante;  
    Motore motoreABenzina;  
}
```

Domanda-1: Quando vengono creati gli oggetti serbatoioCarburante e motoreABenzina?

Risposta: Contestualmente alla creazione dell'oggetto istanza di Automobile!

Relazioni “part-of”

```
class Automobile {  
    // ...  
    Serbatoio serbatoioCarburante;  
    Motore motoreABenzina;  
}
```

Domanda-2: Quando vengono distrutti gli oggetti serbatoioCarburante e motoreABenzina?

Risposta: Contestualmente alla distruzione dell'oggetto istanza di Automobile!

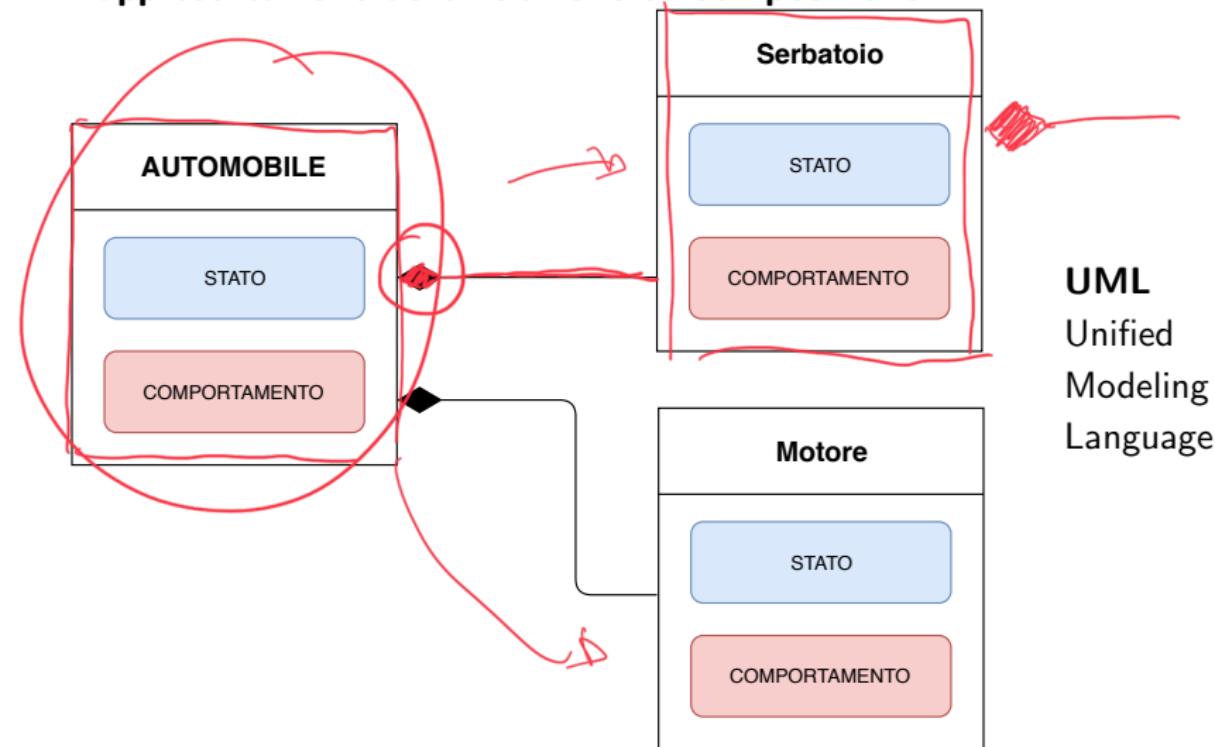
Relazioni “part-of”

La relazione {Automobile,Motore} e
{Automobile,Serbatoio} si dice **Relazione stretta**
o di **Composizione**.

Nella relazione di composizione, l'oggetto *contenuto*
(ES: motore o serbatoio) viene creato e distrutto
insieme all'oggetto *contenitore* (ES: Automobile).

Relazioni “part-of”

Rappresentazione della relazione di Composizione



Relazioni “part-of”

Adesso si consideri la relazione {Automobile,Persona}, in particolare la classe Persona:

```
1 class Persona {  
2     string nome;  
3     //...;  
4     bool allaGuida;  
5     //...;  
6  
7     void setAllaGuida (bool); g  
8     bool getAllaGuida ();  
9 };
```

Durante l'esecuzione dell'applicazione, l'oggetto persona può essere o non essere alla guida dell'automobile (`allaGuida==false` oppure `allaGuida==true`).

Relazioni “part-of”

Infatti:

```
1  Automobile :: Automobile(  
2      Persona* guidatore, ... )  
3  
4  void Automobile :: avviamento () {  
5      guidatore->setAllaGuida(true);  
6      motoreABenzina . accensione ();  
7  }  
8  void Automobile :: spegnimento () {  
9      guidatore->setAllaGuida(false);  
10     motoreABenzina . spegnimento ();  
11 }
```



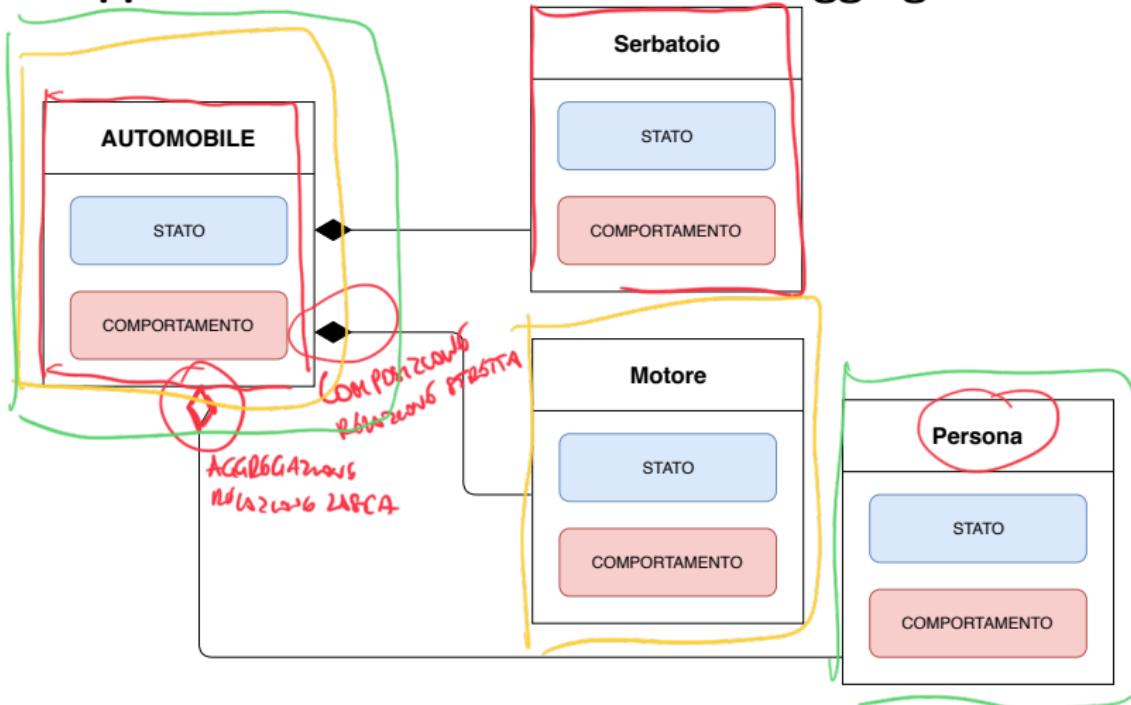
Relazioni “part-of”

La relazione che intercorre tra la classe **Automobile** e la classe **Persona** si dice lasca o anche **Relazione di Aggregazione**.

L'oggetto di tipo **Persona** (oggetto contenuto) ha *vita propria* anche senza l'oggetto contenitore.

Relazioni “part-of”

Rappresentazione della relazione di Aggregazione



Relazioni “part-of”

REMARK. Nella relazione di **composizione**:

- l'oggetto “contenuto” **non ha una vita propria**;
- si realizza con un oggetto “contenuto” **interno al contenitore**;
- loggetto “contenitore” è **responsabile della costruzione e distruzione del contenuto**;
Il coordinatore della composizione deve
 - creare l'oggetto contenitore;
 - fornire eventuali valori per **l'inizializzazione del contenuto tramite costruttore** e/o metodi dell'oggetto **contenitore**;

Relazioni “part-of”

REMARK. Nella relazione di **aggregazione**:

- **ciclo di vita** degli oggetti contenuto e contenitore **indipendenti**;
- si realizza con un oggetto “contenuto” **interno al contenitore**;
- contenitore **non responsabile** della costruzione e distruzione del contenuto;

Il **coordinatore della aggregazione** deve

- creare l’oggetto contenuto;
- creare un contenitore passandogli un **puntatore** allo oggetto **contenuto**;

Relazioni “part-of”

Esempi completi di relazioni **part-of**.

A21_10_auto.cpp

(automobile.cpp, automobile.h)

A21_11_distributore.cpp

(distruzione_bevande.cpp, distributore_bevande.h)

Array di oggetti

Array di frecce.

```
1 const int NUM_FRECCE = 20;
2 Bersaglio b(10);
3 Freccia F[NUM_FRECCE];
4 int tot=0, i=0;
5 while ( i < NUM_FRECCE ){
6     F[i].lancia(b);
7     i++;
8 }
9 i=0;
10 while ( i < NUM_FRECCE )
11     tot += b.punteggio(F[i++]);
```

Array di oggetti

Ogni elemento dello array sarà costituito da un oggetto di tipo Freccia.

Ciò implica la creazione di NUM_FRECCCE oggetti di tipo Freccia.

Di conseguenza, per ogni oggetto creato, **sarà invocato il costruttore di tale oggetto!**

```
1 const int NUM_FRECCCE = 20;  
2 Bersaglio b(10);  
3 Freccia F[NUM_FRECCCE];  
4 // ...
```

Array di oggetti

Altro esempio: la classe **Frazione**.

Il costruttore della classe **Frazione** prevede una lista di argomenti..

```
1 class Frazione {  
2     //...  
3     Frazione(int x, int y);  
4     //...  
5 }
```

...e nel caso in cui si voglia creare una collezione di oggetti **Frazione**?

Array di oggetti

Allora si può invocare esplicitamente il costruttore di tutti gli oggetti della collezione.

ut `num[3] = {1,2,3};`

ma così

```
1 Frazione frazioni[3] = {Frazione(1,2), Frazione(3,4) \
2 , Frazione(4,5)};
3 //OPPURE
4 Frazione frazioni[3] = {{1,2}, {3,4}, {4,5}};
```

Nella seconda forma, il compilatore usa ogni coppia di interi come argomento per il costruttore della classe `Frazione`, operando di fatto una **conversione** da tipi primitivo a tipo `Frazione`.

Array di oggetti

Altra soluzione: usare un **array di puntatori e allocare dinamicamente gli oggetti.**

```
1 Frazione *frazioni[10];
2
3 for(int i=0; i<9; i++)
4     frazioni[i] = new Frazione((i+1)*5, i+2);
```

Array di oggetti

NOTA

```
1 Frazione *ptr = new Frazione(1,2);
2 ptr->numeratore(); //invocaz. metodo
3 //OPPURE
4 (*ptr).numeratore();
```

Se `ptr` è un puntatore ad un oggetto che espone un metodo `foo()`, per l'invocazione del metodo attraverso il puntatore si può:

- usare la sintassi della linea 2 (operatore “freccia”);
- ... oppure la sintassi classica (linea 3) dereferenziando preventivamente il puntatore.

Array di oggetti

Esempi svolti

A22_01.cpp

A22_02.cpp

A22_03.cpp

A22_04.cpp

FINE