C++ · Oggetti e classi

Fondamenti
di informatica

Michele Tomaiuolo
tomamic@ce.unipr.it
http://www.ce.unipr.it/people/tomamic

Funzione

- Operatore, applicato a operandi, x calcolare risultato
  - Occorre specificare il tipo di parametri e risulato
  - Nel corpo: `return` per terminare e restituire un risultato
  - `Type functionName(Type paramName, Type paramName…) { /* … */ }`

```
int somma(int m, int n) {
  int result = m + n;
  return result;
}

void fun() { /* procedure */ }
```

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Funzioni utente

```
int readNumber() {
  int n;
  cout << "Insert number: ";
  cin >> n;
  return n;
}

int max (int m, int n) {
  int result = m;
  if (n > m) { result = n; }
  return result;
}

int main () {
  int a = readNumber();
  int b = readNumber();
  cout << "The maximum is " << max(a, b) << endl;
  return 0;
}
```

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

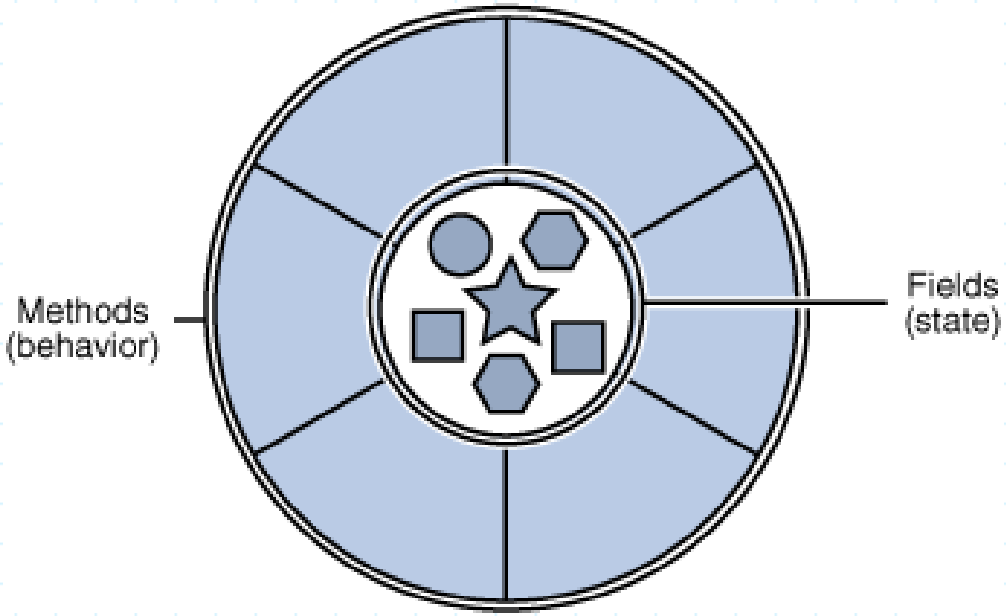
Parametri di funzioni

- Parametri formali
  - Nomi usati nella *definizione*: completamente arbitrari
- Parametri effettivi
  - Ciò che viene passato alla *chiamata*
- C++: parametri normalmente passati per valore
  - Variabili passate come param. effettivi: non modificate!
- Array passati per riferimento
  - Array = indirizzo in memoria del primo elemento
  - Modifica in funzione su array: permanente!
  - Bisogna passare anche la dimensione

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

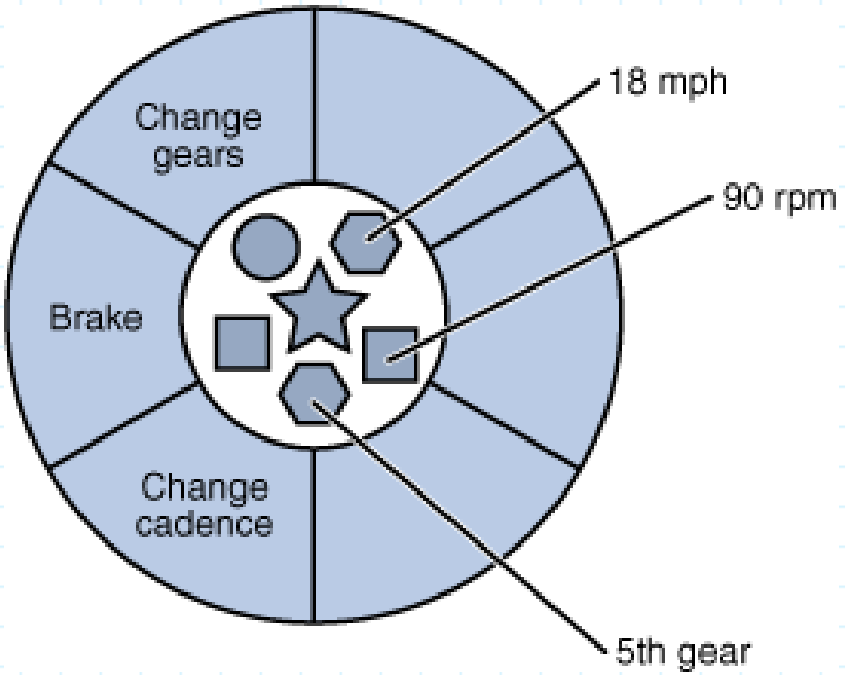
Oggetto

- Astrazione per un *oggetto fisico* o un *concetto* (in problema, o soluzione) caratterizzato da:
  - Uno stato **interno**
  - Un insieme di servizi **esposti** agli altri oggetti
  - Una identità



Esempio: una bicicletta

- Ha uno stato: marcia innestata, frequenza di pedalata, velocità attuale
- Espone dei servizi: cambiare marcia, cambiare frequenza di pedalata, azionare i freni
- Ha una identità:
  # telaio,
  posizione fisica



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Classi di oggetti software

- Nei linguaggi class-based (come Java e C++)...
  - Ogni oggetto ha (almeno) una classe origine
  - È descritto tramite gli attributi e i metodi della classe (insieme, famiglia) di cui fa parte
- Poi, creazione oggetti e invio messaggi
- Istanza della classe X ≡ oggetto di classe X
  - Ogni istanza ha uno stato e una identità (locazione in memoria) distinti...
  - Anche rispetto alle istanze della stessa classe

Bicycle: progetto C++

- C++: definizione della classe separata dalla definizione (implementazione) dei suoi metodi
  - Definizione fornita agli utenti
  - Implementazione compilata in libreria
- Sorgenti organizzati in 3 file:
  - `bicycle.h` – definizione della classe
  - `bicycle.cpp` – implementazione dei metodi
  - `main.cpp` – applicazione che usa la classe
  - Dall'ambiente di sviluppo: *Add new* → *C++ Class*

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

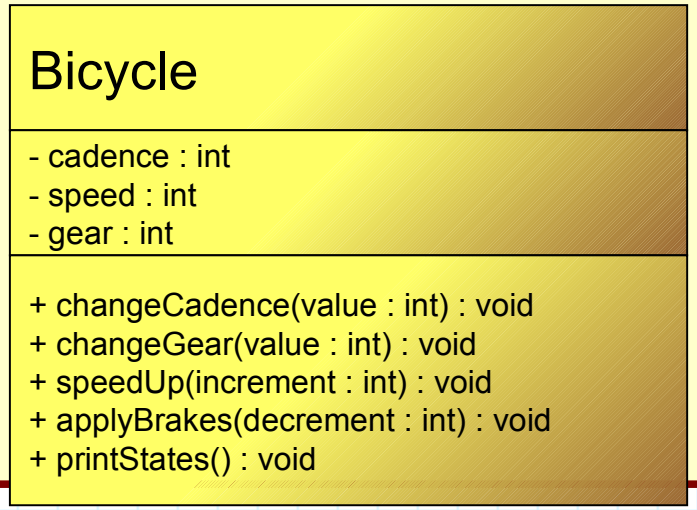
Definizione: bicycle.h

```
#ifndef BICYCLE_H
#define BICYCLE_H

class Bicycle {
public:
  Bicycle();
  Bicycle(int c, int g, int s);
  void changeCadence(int c);
  void changeGear(int g);
  void speedUp(int inc);
  void applyBrakes(int dec);
  void printStates();
private:
  int cadence;
  int speed;
  int gear;
};
#endif
```

Metodi nella sezione “public”

Stato nella sezione “private”
I metodi possono accedervi



Implementazione: bicycle.cpp

```
#include "bicycle.h"

Bicycle::Bicycle() { cadence = 0; gear = 0; speed = 0; }

Bicycle::Bicycle(int c, int g, int s) {
  cadence = c; gear = g; speed = s;
}

void Bicycle::changeCadence(int c) { cadence = c; }

void Bicycle::changeGear(int g) { gear = g; }

void Bicycle::speedUp(int inc) { speed += inc; }

void Bicycle::applyBrakes(int dec) { speed -= dec; }

void Bicycle::printStates() {
  cout << "cadence: " << cadence << " speed:"
    << speed << " gear:" << gear;
}
```

Nomi dei metodi preceduti dal nome della classe

Applicazione: main.cpp

```
#include "bicycle.h"

int main() {
  // Create two different Bicycle objects
  Bicycle bike1;
  Bicycle bike2;

  // Invoke methods on those objects
  bike1.changeCadence(50);
  bike1.speedUp(10);
  bike1.changeGear(2);
  bike1.printStates();

  bike2.changeCadence(40);
  bike2.speedUp(20);
  bike2.changeGear(3);
  bike2.printStates();
}
```

Costruttori

- La creazione di un oggetto deve coincidere con la sua inizializzazione ad uno stato coerente
- Il costruttore è un metodo speciale
  - Nessun tipo di ritorno
  - Stesso nome della classe
  - Prevista una “sezione di inizializzazione”, prima del corpo
  - Eseguito automaticamente alla creazione
- Dichiarazione: `Bicycle b(40, 1, 20);`
  - Ha l'effetto di allocare l’oggetto in memoria...
  - … e di eseguire il costruttore per inizializzarlo

Metodi getter e setter

- A volte si introducono metodi che consentono di manipolare esplicitamente lo stato dell'oggetto
  - `void Bicycle::setSpeed(int s) { speed = s; }`
  - `int Bicycle::getSpeed() { return speed; }`
- Getter e setter dovrebbero essere minimizzati
  - Oggetto: ha responsabilità di gestire il suo stato
  - Deve offrire tutti i servizi utili, basati sul suo stato

I metodi getter e setter sono utili soprattutto se non si limitano a leggere o scrivere variabili

Creazione e uso di oggetti

- Oggetto nello stack (allocaz. decisa a *compile-time*)
  - Ciclo di vita gestito automaticamente, limitato allo *scope*
- Oggetto nello heap (allocaz. decisa a *run-time*)
  - Ciclo di vita gestito esplicitamente con `new` e `delete`

```
Bicycle b(40, 1, 20); // automatic memory, stack
b.printStates();

Bicycle* bike = new Bicycle(40, 1, 20); // heap
bike->printStates();
/* … */
delete bike;
```

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

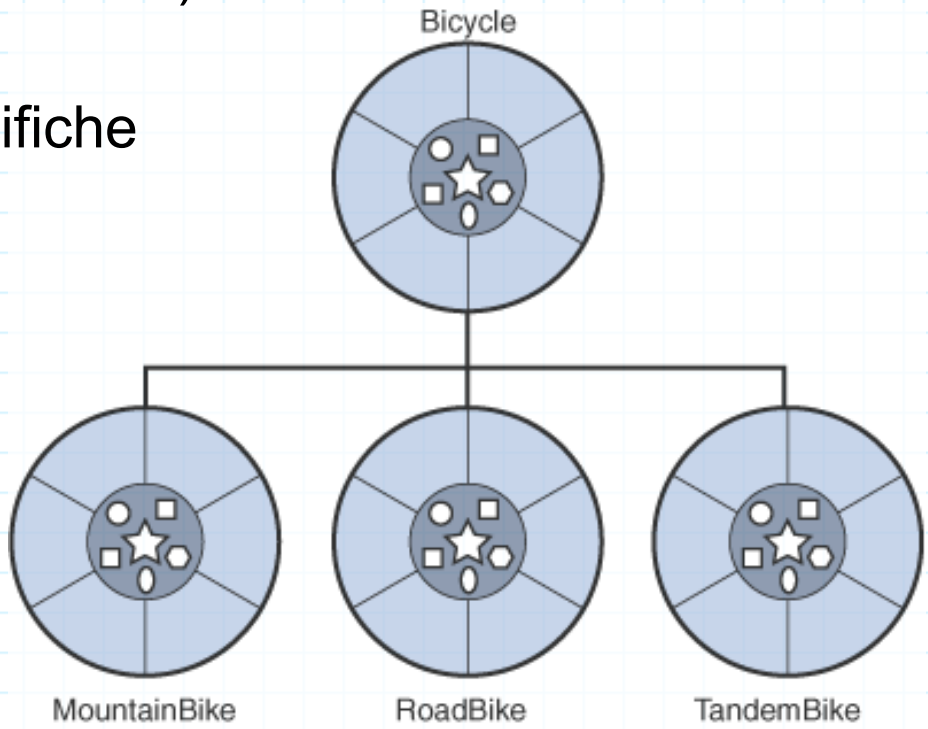
Ereditarietà

- Tra le classi è possibile definire una relazione di sotto-classe (sotto-insieme)
  - Classi *Animale*, *Felino*, *Gatto*
  - *Gatto* sotto-classe di *Felino* (classe derivata diretta)
  - *Gatto* sotto-classe di *Animale* (classe derivata indiretta)
  - *Animale* classe base di *Felino* e di *Gatto*
- Classi esistenti: base per creare nuove classi

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Tipi di biciclette

- Diversi tipi di biciclette: modellati come sotto-classi di Bicycle
  - Condividono caratteristiche di Bicycle (velocità, marcia, cadenza)
  - Aggiungono caratteristiche specifiche (# corone, forma manubrio, # sellini)



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Metodi e campi ereditati

- La sotto-classe...
  - Eredita tutte le caratteristiche `public` della classe base
  - Non può accedere alle caratteristiche `private` della classe base
  - Può definire nuove caratteristiche che non sono presenti nelle classi base (Eckel: is-like-a)
- La classe base...
  - Può definire delle caratteristiche `protected`, a cui solo lei e le sotto-classi possono accedere

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Ereditarietà in C++

- Classe derivata costruita a partire da classe Base
  - ```
    class MountainBike : public Bicycle {
      // additional features
    }
    ```
- Ereditarietà multipla tra le classi C++
  - Una classe può avere più classi base *dirette*
  - Java: ereditarietà singola + interfacce
- Non esiste una classe base da cui tutte le classi implicitamente derivano
  - Java: `Object`, base di tutte le classi

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/
    ```