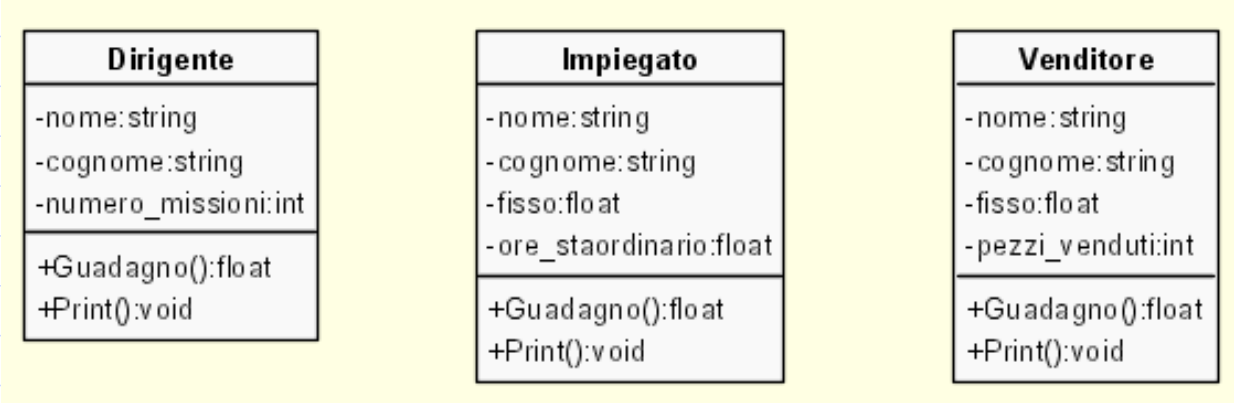
Astrazione

- Classi Astratte
- Classi Interfaccia

Classe astratta: *premessa 1*

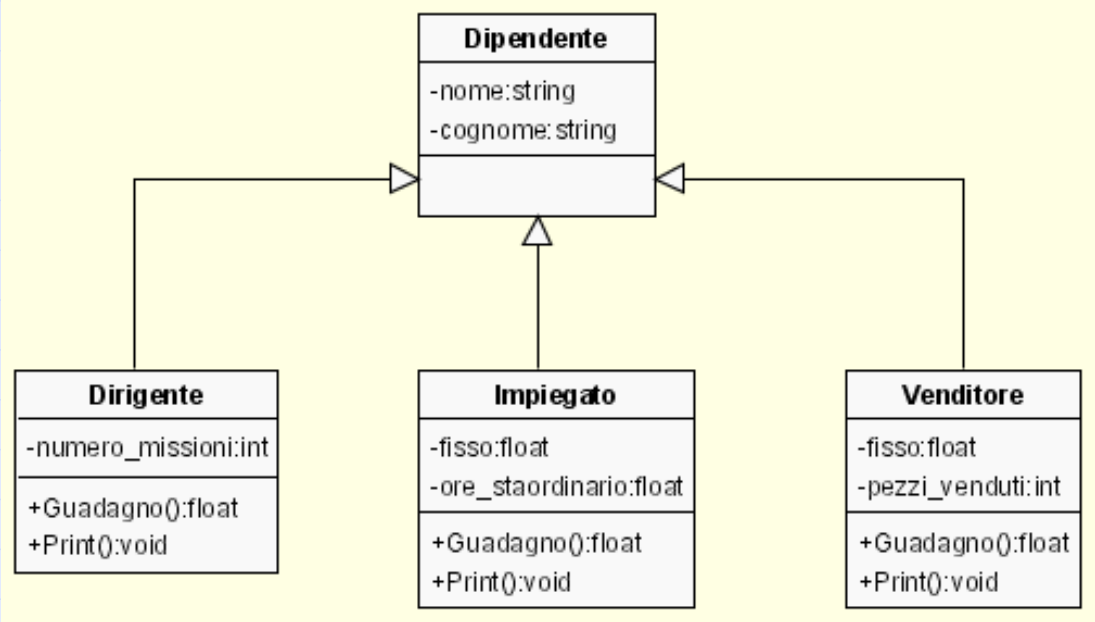
- Supponiamo si debbano progettare le seguenti 3 classi per un'applicazione aziendale ...



- Osservazione: le 3 classi hanno attributi e metodi in comune. Come ridurre la complessità del progetto?

Classe astratta: *premessa 2*

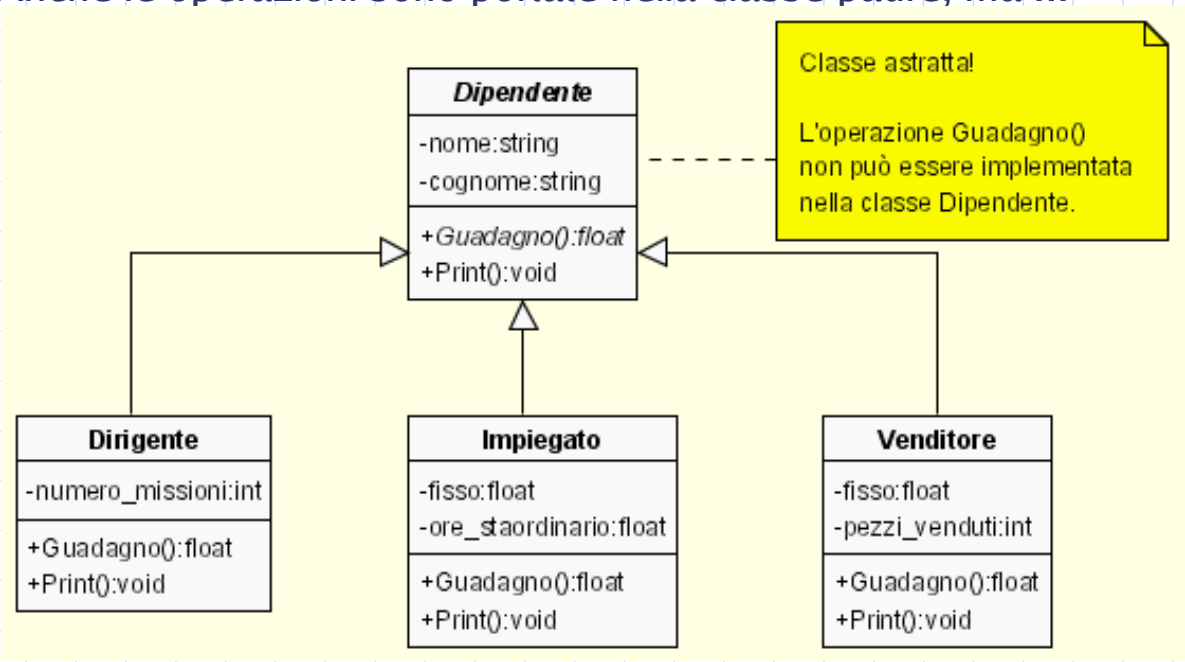
- L'ereditarietà permette di semplificare ...



- Come comportarsi per le operazioni comuni? Purtroppo, queste hanno implementazioni differenti nelle 3 classi!!

Classe astratta: *premessa 3*

- Anche le operazioni sono portate nella classe padre, ma ...



- Dipendente::Print() stampa solo nome e cognome
- Dipendente::Guadagno() non ha implementazione !!!

Classe astratta: *definizione*

- Una classe **A** è astratta se possiede almeno un'operazione astratta.
- Un'operazione è astratta quando non è possibile definirne un'implementazione.
- Da una classe astratta non è possibile istanziare oggetti.
- Se **A** è astratta e **B** eredita da **A**, allora **B** deve fare l'overriding delle operazioni astratte (non implementate) di **A**.
- Le classi astratte sono molto importanti per la progettazione. Una classe astratta **A** serve per stabilire proprietà di classi concrete (ottenute per ereditarietà da **A**).

Classi Astratte in C++

In C++ si può definire un metodo virtual puro mediante la seguente notazione:

```
class Dipendente {
public:
   Dipendente( …… );
        :
virtual double guadagno() const = 0;
virtual void print() const;
        :
private:
        :
};
```

Metodo virtual puro

Metodo virtual

La presenza di un metodo **virtual puro** rende la classe **astratta**.

Classi astratte in Java

- In Java una classe che non richiede istanze può essere dichiarata **abstract**

```
public abstract class ImpiegatoGenerico {
   //…
}
```

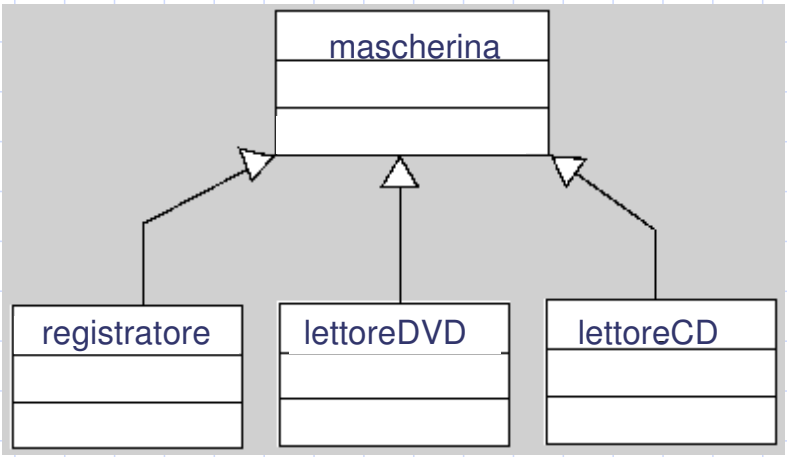
- Una classe astratta può contenere tutto ciò che ha una classe normale più **metodi astratti** (metodi senza implementazione) dichiarati tramite il modificatore **abstract.**

```
abstract double CalcolaStipendio();
    //nessuna implementazione!
```


Ancora sulle classi astratte in Java

- Un metodo può essere dichiarato **abstract** solo se lo è la classe
- Le classi astratte possono contenere costruttori che le classi derivate possono richiamare

Classi interfaccia: *premessa*



- A volte una serie di classi hanno operazioni comuni
- Tutte le operazioni hanno però implementazioni differenti
- La classe astratta che le rappresenta ha allora ha tutti i metodi virtual puri (è il caso della classe `Mascherina` con le operazioni `avanti(), stop(), indietro(),` etc., comuni a tutte le classi derivate)

Classi interfaccia: *definizione*

- Una classe **A** è un'**interfaccia** se possiede solo operazioni astratte.
- Un'interfaccia è quindi una classe astratta.
- Si usa per dire cosa *è* un oggetto rispetto a cosa *fa*.
- Da una classe interfaccia non è possibile istanziare direttamente oggetti ma solo derivare classi da cui poi istanziare oggetti.
- Se **A** è interfaccia e **B** eredita da **A**, allora **B** deve fare l'overriding di **tutte** le operazioni di **A**.

Classi Interfaccia in C++

In C++ non esistono parole chiave per definire un'interfaccia.
Si realizza un'interfaccia mediante una serie di metodi virtuali puri.

```
class Mascherina {
public:
virtual void avanti() const = 0;
virtual void stop() const = 0;
virtual void indietro() const =0;

};
```

Metodi virtual puri

Interfacce

- Un'interfaccia in Java è una raccolta di comportamenti astratti
- Un'interfaccia Java non contiene altro che metodi astratti (senza implementazione) e costanti
- Es:

Nel package java.util :

```
interface Sferico {
    void lanciare();
    void ruotare();
  //…
}
```

```
interface Comparable {
    int compareTo(Object);
}
```

Classi e interfacce

- Una classe che implementa una interfaccia definisce tutti i metodi dell'interfaccia stessa. Es:

```
public class elementoLista implements Comparable {
  int valore;    //attributo
  elementoLista(…) {  //costruttore
  …}
  …      //altri metodi
 public int compareTo(Object obj){
     elementoLista temp = (elementoLista)obj; //casting
     if (this.valore < temp.valore) return 1;
     if (this.valore > temp.valore) return –1;
     else return 0;
  }
}
```


Interfacce ed ereditarietà singola in Java

- Una gerarchia di classi ad ereditarietà singola è limitante specialmente quando si è in presenza di un comportamento che deve essere utilizzato da classi in rami diversi dell'albero di derivazione.
- L'ereditarietà multipla è una soluzione al problema. Costo: il linguaggio è più complesso
- Java vuole essere semplice: adotta l'ereditarietà singola, ma offre i costrutti **interface (interfaccia)** per ovviare ai problemi del primo punto.
- Infatti in Java una classe può impementare più di una interfaccia