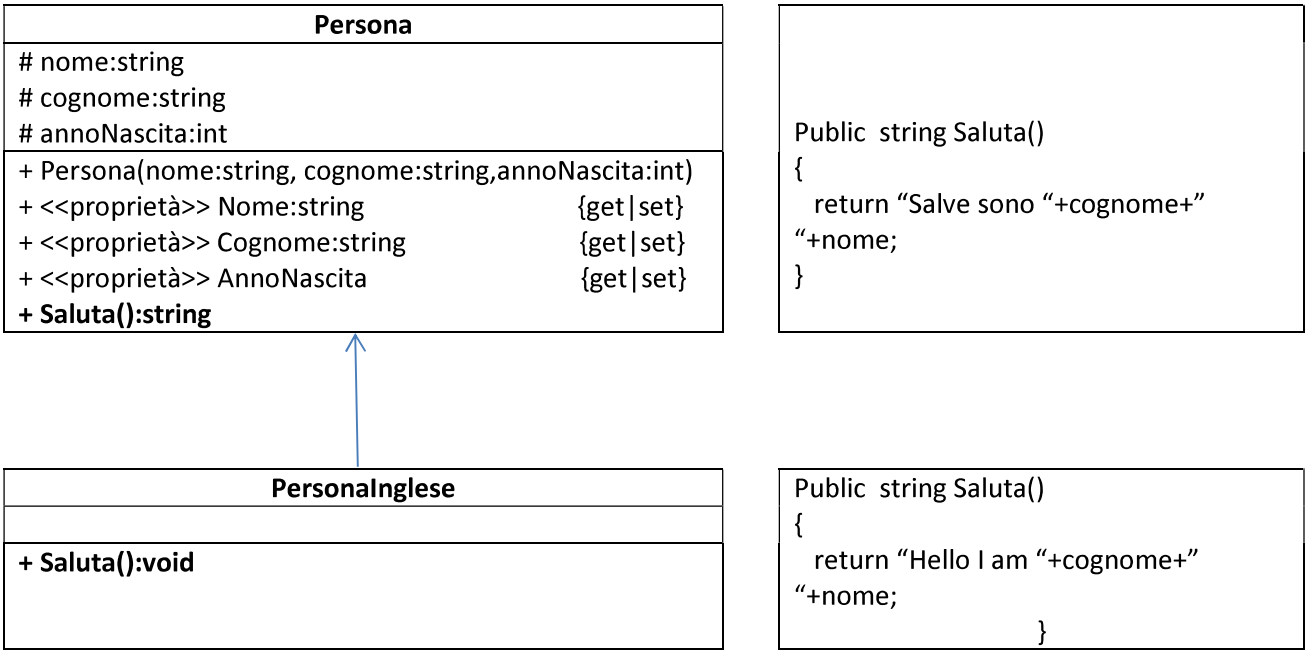
Consideriamo la seguente gerarchia di classi:



Qui siamo in presenza di ridefinizione (overriding) del metodo Saluta().

Nel codice consumer se io scrivo:

Persona p=new Persona("Mario","Rossi",2000);
p.Saluta() → Salve sono Rossi Mario

Persona p2=new PersonaInglese("William","Shakespeare",1564);
p2.Saluta() → Salve sono William Shakespeare

Se voglio che p2 si comporti come PersonaInglese devo :

- dichiarare virtual il metodo Saluta() nella classe Persona
- dichiarare override il metodo Saluta() nella classe PersonaInglese

OVERRIDING E OVERLOADING

Attenzione a non confondere :

• il sovraccarico dei metodi (**overloading**)

situazione in cui oltre al corpo del metodo è differente anche la sua firma

L'overloading consente di definire in una stessa classe **più metodi aventi lo stesso nome**, ma che **differiscano nella firma**, cioè nella sequenza dei tipi dei parametri formali.

Per l'overloading è il compilatore che determina quale dei metodi verrà invocato, in base al numero e al tipo dei parametri attuali.

• con la ridefinizione (**overriding**)

situazione in cui la firma del metodo è identica ma è differente il corpo

BINDING

Si chiama binding (“bind” significa “legare”) il meccanismo che determina quale metodo deve essere invocato in base alla classe di appartenenza dell'oggetto

• Si dice che si ha binding statico (**early binding**)

quando il metodo da invocare viene determinato in fase di **compilazione**

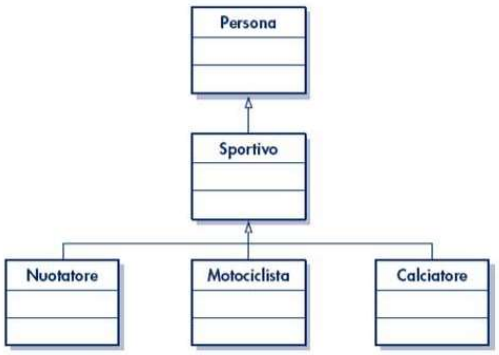
• Si dice che si ha binding dinamico (**late binding**)

quando il metodo viene determinato durante l'esecuzione (a **run time**)

consente di ridefinire un metodo in una sottoclasse: il metodo originale e quello che lo ridefinisce hanno necessariamente la stessa firma, e solo a tempo di esecuzione si determinerà quale dei due deve essere eseguito se nella classe base il metodo è **virtuale**.

GERARCHIA DI CLASSI

L’ereditarietà può estendersi a più livelli generando quindi una gerarchia di classi.



• Una classe derivata può, a sua volta, essere base di nuove sottoclassi

• Sportivo è sottoclasse di Persona ed è classe genitore di Nuotatore, Motociclista e Calciatore

• Nella parte alta della gerarchia troviamo le classi generiche, scendendo aumenta il livello di specializzazione

CLASSI ASTRATTE

Man mano che si sale nella gerarchia dell’ereditarietà, le classi diventano sempre più generiche e quindi più astratte.

• Ad un certo punto la classe di livello superiore diventa a tal punto generica che la si può pensare come una base (classe astratta) per costruire le altre classi e non utilizzabile direttamente

Va usata la parola chiave abstract :

- Per dichiarare la **classe abstract**
- Usata anche per dichiarare i **metodi abstract** (solo definiti non implementati)

– Tipicamente le classi abstract contengono uno o più metodi abstract

– Tutte le sottoclassi concrete **DEVONO** fare l'override dei metodi astratti

ERDITARIETà SINGOLA E MULTIPLA

In genere nella progettazione OO sono possibili due tipi di ereditarietà:

– ereditarietà singola

– ereditarietà multipla

• L'ereditarietà singola impone ad una sottoclasse di derivare da una sola classe genitore

• L'esempio presentato precedentemente è un caso di ereditarietà singola: ogni sottoclasse ha una sola classe base, mentre è possibile da una classe genitore avere più classi derivate

• Vari linguaggi ad oggetti pongono il vincolo dell'ereditarietà singola per problemi di chiarezza e semplicità d'implementazione, C# è uno di questi.

• Quindi non è possibile una definizione di classe del tipo:

class A : B,C

C# per implementare l'ereditarietà multipla utilizza le interfacce

ENTITÀ ASTRATTE

Moltissime entità che usiamo per descrivere il mondo non sono reali

•sono pure **categorie concettuali**, ma sono comunque utili per esprimersi

ESEMPIO: GLI ANIMALI

•parlare di "animali" ci è molto utile, ma a ben pensarci non esiste "il generico animale"!

•nella realtà esistono solo animali specifici

– cani, gatti, lepri, serpenti, pesci, …

Da qui nasce il concetto di **CLASSE ASTRATTA**

•una classe che rappresenta una categoria concettuale astratta , ad esempio, Animale

•di cui quindi non possono esistere istanze perché non esistono "animali qualsiasi"

•Le istanze apparterranno invece a sue sottoclassi concrete , ad esempio Cane, Gatto, Corvo, …

Una CLASSE ASTRATTA potrebbe essere realizzata da una normale classe , con la convenzione di non crearne mai istanze

•ma è meglio poterla qualificare esplicitamente , in modo che la sua natura astratta sia evidente

e che il compilatore possa verificare l'assenza di istanze

•Per questo C# introduce la parola chiave **abstract** per etichettare

- sia la classe in quanto tale
- sia uno o più metodi (metodi astratti)

UN ESEMPIO:

```
public abstract class Animale
{
        public abstract string siMuove();
        public abstract string vive();
}
```

Esprime il fatto che ogni animale si muove in qualche modo e vive da qualche parte, ma in generale non si può dire come, perché varia da un animale all'altro.

Tecnicamente:

•i metodi astratti non hanno corpo, c'è solo un ";"

•se anche solo un metodo è abstract, la classe intera dev'essere abstract altrimenti, ERRORE

La classe astratta è un potente strumento per modellare gli aspetti comuni di molte realtà

Operativamente, una classe astratta:

•lascia "in bianco" uno o più metodi, dichiarandoli senza però definirli

•tali metodi verranno prima o poi implementati da qualche classe derivata (concreta)

- per essere concreta, una classe deve disporre di una implementazione per tutti i metodi ex-astratti.
- se ne implementa solo alcuni, rimane astratta.

UN ESEMPIO COMPLETO

Completiamo il mini-esempio precedente:

•ogni animale risponde a due metodi che restituiscono una stringa descrittiva:

- vive() indica DOVE VIVE l'animale
- siMuove() indica COME SI MUOVE l'animale

•il metodo ToString() indipendente dallo specifico animale che restituisce i dati dell'animale

•tutti gli animali hanno la stessa rappresentazione interna, data dal loro nome.

```
public abstract class Animale
{
        private string nome;
        public Animale(String s) { nome=s; }
        public abstract string siMuove();
        public abstract string vive();
        public override string ToString()
        {
```

```
        return nome+ ", " + ", si muove" + siMuove()+ " e vive " + vive());
}
}

public class Uccello : Animale
{
        int grandezzauova;
        ……
        public override string siMuove() {return “volando”;}
        public override string vive() { return “in aria”;}


}
public class Pesce : Animale
{
        int lunghezza;
        ……
        public override string siMuove() {return “nuotando”;}
        public override string vive() { return “in acqua”;}


}
```