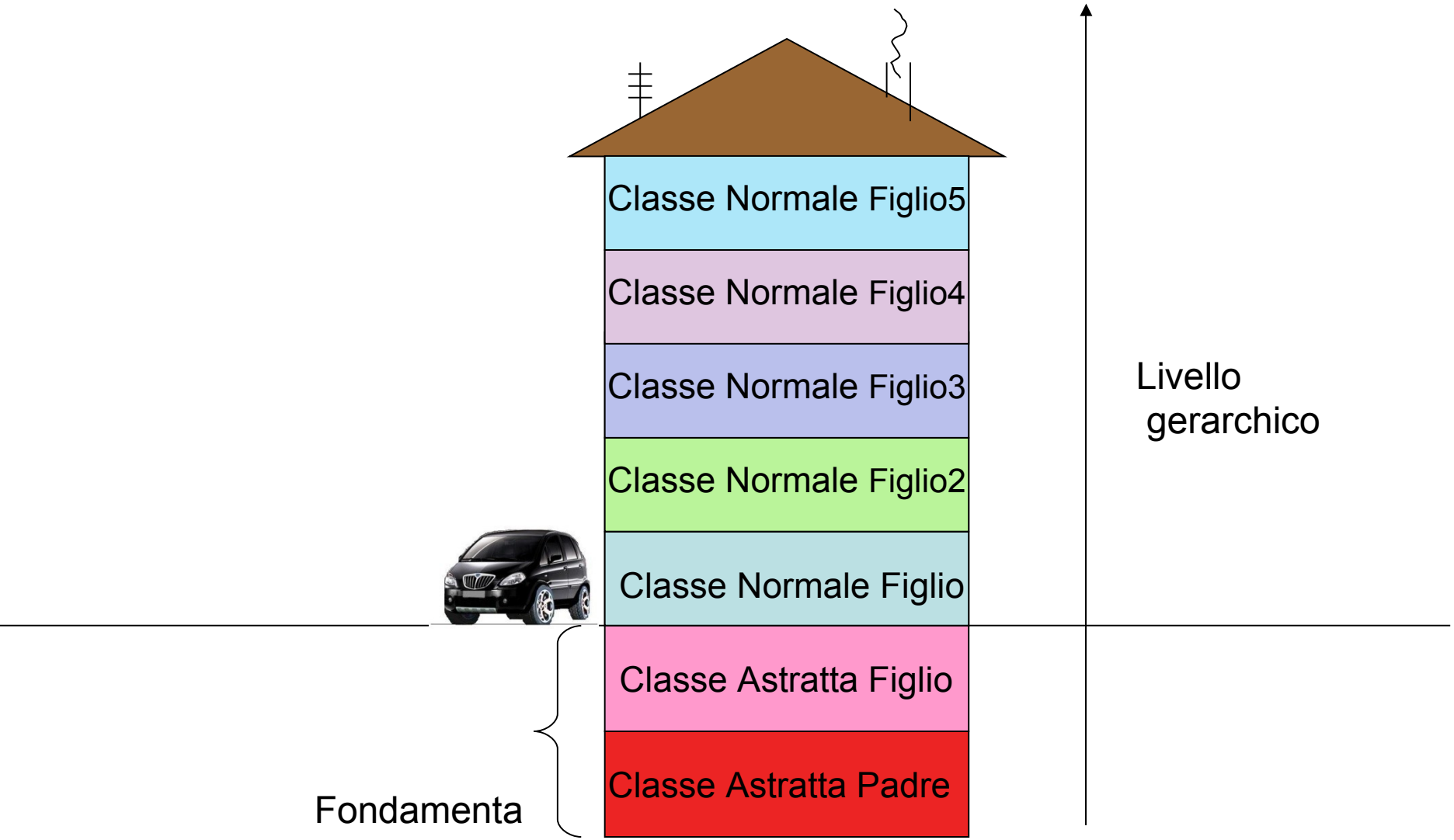
# Classi Astratte
# vs
# Classi
# vs
# Interfacce

A cura del docente Giuliano Pellegrini Parisi - © 2009

Una *abstract class* o classe astratta rappresenta la piattaforma di base, le fondamenta da cui iniziare a costruire una gerarchia di classi assicurando all' intera struttura l'immutabilità delle proprie caratteristiche nel tempo e garantendo così uno standard duraturo nel tempo ad ogni evenienza.
Una *abstract class* può essere vista come un contratto tra la classe astratta e l'intera struttura gerarchica, nel quale la *abstract class* si impegna a mantenere identico nel tempo lo standard di progettazione stabilito dall'analista.

Per comprendere il concetto logico di classe astratta pensate alle fondamenta di una casa, le quali vengono progettate una sola volta e poi restano durature nel tempo. Nell'eventualità che si debbano rivedere i parametri di progettazione delle fondamenta, diviene obbligatorio distruggere la casa e sistemare le fondamenta.
Discorso analogo avviene nelle classi astratte, se l'analista ha errato la progettazione della o delle classi astratte, l'intera struttura gerarchica deve venire rivista.


Classe Normale Figlio5
Classe Normale Figlio4
Classe Normale Figlio3
Classe Normale Figlio2
Classe Normale Figlio
Classe Astratta Figlio
Classe Astratta Padre
Livello gerarchico
Fondamenta

Si deve utilizzare la parole chiave “abstract” prima della definizione della classe. Si valuti se poi rendere pubblica la classe astratta.

```
abstract class Capostipite
{
    ...
    ...
}
```

```
abstract public class Capostipite
{
    ...
    ...
}
```

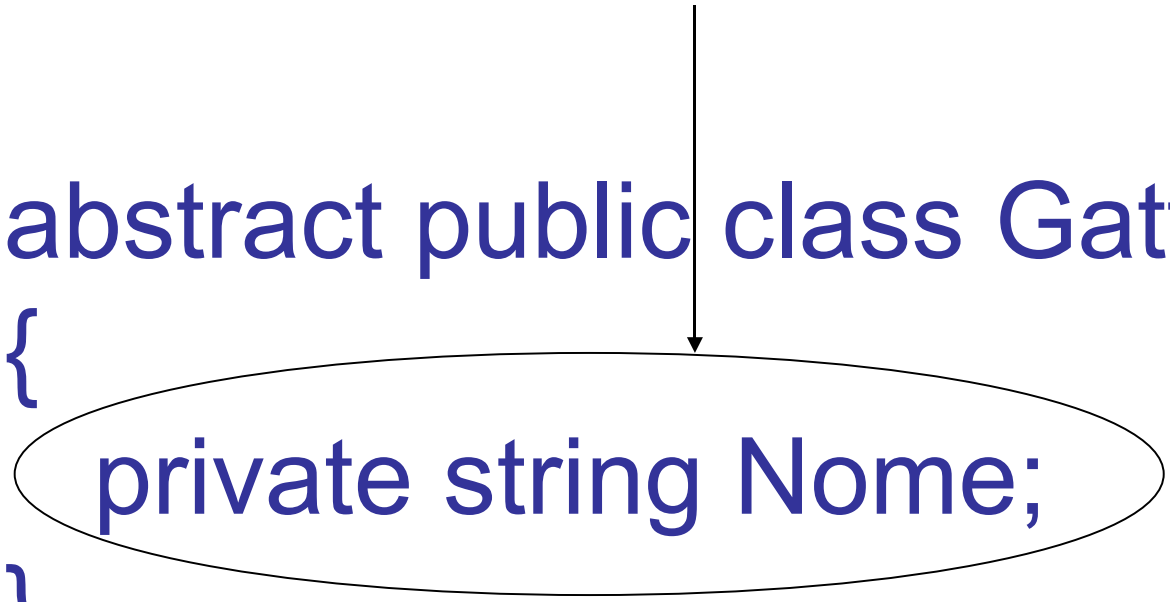
Che tipo di informazioni può contenere una *abstract class*?

Una *abstract class* può contenere:

- dati membro
- costruttori di classe
- metodi get e set
- metodi astratti ma solo definiti

Vediamo in dettaglio i vari casi!

Dato membro private che rappresenta il nome del gatto.
Vale la stessa logica che avete imparato per le classi normali



```
abstract public class Gatto
{
    private string Nome;
}
```

Definizione del costruttore di classe Gatto con il quale si inizializza il dato membro private come avete imparato per le classi normali.

```
abstract public class Gatto
{
   private string Nome;

   public Gatto(string Nome)
   {
     this.Nome=Nome;
   }
}
```

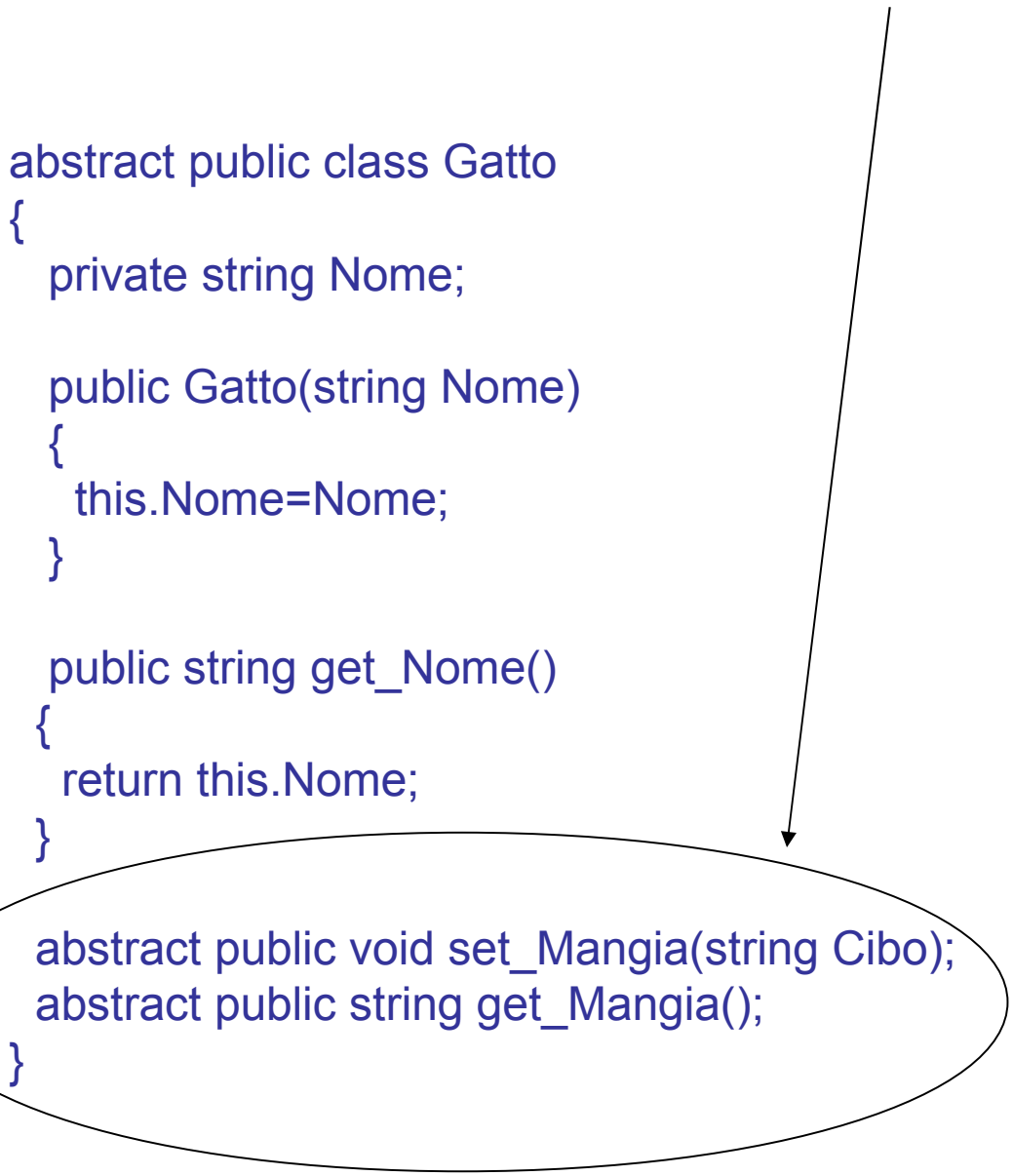
Definizione di un metodo get per la lettura del dato membro private di tipo string come avete imparato per le classi normali

```
abstract public class Gatto
{
   private string Nome;

   public Gatto(string Nome)
   {
     this.Nome=Nome;
   }

   public string get_Nome()
   {
     return this.Nome;
   }
}
```

Definizione di due metodi abstract solo definiti ma non implementati.



```
abstract public class Gatto
{
   private string Nome;

   public Gatto(string Nome)
   {
     this.Nome=Nome;
   }

   public string get_Nome()
  {
    return this.Nome;
  }

  abstract public void set_Mangia(string Cibo);
  abstract public string get_Mangia();
}
```

La definizione dei dati membro,del costruttore e dei metodi get e set sono quelli che avete studiato ed imparato per le classi normali.

I metodi abstract possono venire definiti solo all'interno di una classe abstract ma NON implementati all'interno della classe astratta o di una classe figlia anch'essa abstract.

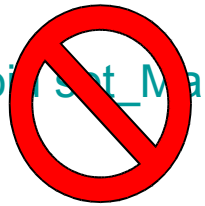
```
abstract public class Gatto
{
   …

  abstract public void set_Mangia(string Cibo);
  abstract public string get_Mangia();
}

abstract public class Persiano : Gatto
{
  …

  public void set_Mangia(string Cibo)
  {
    …
  }

  public string get_Mangia()
  {
    …
  }
}
```



La classe astratta Persiano è figlia della classe astratta Gatto

E' VIETATO implementare metodi astratti all'interno di una classe astratta, indipendentemente dal fatto che sia classe astratta figlia o classe astratta padre.

```
abstract public class Gatto
{
   …

  abstract public void set_Mangia(string Cibo);
  abstract public string get_Mangia();
}

public class Persiano : Gatto
{
  …

  public void set_Mangia(string Cibo)
  {
    …
  }

  public string get_Mangia()
  {
    …
  }
}
```



La classe Persiano
è figlia della classe astratta
Gatto

E' OBBLIGATORIO implementare i metodi astratti all'interno di una classe figlia che eredita una classe astratta con metodi astratti

La prima classe che eredita un classe astratta con metodi astratti,è OBBLIGATA ad implementare il codice dei metodi astratti perché tutti i metodi *abstract* rappresentano un contratto che la classe astratta ha stipulato con l'intera struttura gerarchica e quindi siete vincolati a rispettare tale contratto nella prima classe figlia che eredita la *abstract class*.

Ma in sostanza quale è il vero impiego delle classi astratte in un progetto OOP?

L’impiego delle *abstract class* non può essere ricondotto ad una ricetta di cucina,ossia non potete decidere il loro impiego sulla base di modelli preconfezionati come il più delle volte viene presentato in libri e/o manuali di settore. L’uso di queste classi dipende esclusivamente dal tipo di struttura gerarchica che andrete a realizzare.

Il nome *abstract* è stato dato proprio per indicare una tipologia di progettazione basata sull'astrazione della realtà che dovete descrivere nella vostra struttura gerarchica.
Cerchiamo di evitare il nozionismo filosofico informatico e di vedere quando impiegare una *abstract class*.

Prima di vedere in quali situazioni adottare le classi astratte diviene fondamentale elencare le importantissime differenza che esistono tra:

- classe
- classe astratta
- interfaccia

| Classe | Classe astratta | Interfaccia |
|---|---|---|
| La classe può essere istanziata nello HEAP tramite l'operatore di allocazione dinamica "new", producendo un oggetto | Le classi astratte NON sono istanziabili nello HEAP | Le interfacce NON sono istanziabili nello HEAP |
| La classe nel linguaggio C# può implementare un eredità di tipo singola | La classe astratta nel linguaggio C# può implementare un eredità di tipo singola | L'interfaccia nel linguaggio C# può implementare un eredità di tipo multipla verso le stesse interfacce |
| La classe può ereditare una sola classe e contemporaneamente infinite interfacce | La classe astratta può ereditare una sola classe astratta e contemporaneamente infinite interfacce | L'interfaccia può ereditare una o più interfacce |
| La classe può ereditare una sola classe astratta e contemporaneamente infinite interfacce | La classe astratta NON può ereditare nessuna classe, ma solo una classe astratta padre ed una o più interfacce | L'interfaccia non può ereditare classi o classi astratte ma solo interfacce |

| Classe | Classe astratta | Interfaccia |
| --- | --- | --- |
| In definitiva possiamo riassumere i seguenti concetti: | | |
| La classe può ereditare una sola classe oppure una sola classe astratta e contemporaneamente infinite interfacce | La classe astratta può solo ereditare una ed una sola classe astratta e contemporaneamente infinite interfacce | L'interfaccia può ereditare una o più interfacce ma nessuna classe o classe astratta |

| Classe | Classe astratta | Interfaccia |
|---|---|---|
| Le classi non possono dichiarare metodi (procedure e/o funzioni) di tipo “abstract”. | Le classi astratte possono dichiarare metodi (procedure e/o funzioni) di tipo “abstract”. | Le interfacce non possono dichiarare metodi (procedure e/o funzioni) di tipo “abstract”. |
| Le classi possono implementare metodi (procedure e/o funzioni) di tipo “abstract”. | Le classi astratte non possono implementare metodi (procedure e/o funzioni) di tipo “abstract”. | Le interfacce non possono implementare metodi (procedure e/o funzioni) di tipo “abstract”. |
| Le classi possono avere dati e funzioni membro (metodi) e costruttori. | Le classi astratte possono avere dati e funzioni membro (metodi) e costruttori. | Le interfacce non possono avere costruttori, ne dati membro, ma solo la dichiarazione di funzioni membro (metodi) e non la loro implementazione |

| Classe | Classe astratta | Interfaccia |
|---|---|---|
| La classe rappresenta una struttura logica nella quale l'analista organizza la struttura gerarchica la quale potrebbe subire delle variazioni nel tempo. | La classe astratta rappresenta un livello di astrazione in fase di progettazione superiore alla classe e serve per definire uno standard che resterà uguale e duraturo nel tempo senza mai subire variazioni. | L'interfaccia rappresenta un livello di astrazione che riguarda i casi particolari o comunque delle situazioni che si verificano in rari casi e che posso accumunare diverse classi o classi astratte. |
| Da un punto di vista implementativo e funzionale, la classe non stipula nessun contratto con la struttura gerarchica,ecco perché può subire se necessario delle modifiche nel tempo. | Da un punto di vista implementativo e funzionale, la classe astratta stipula un contratto con l'intera struttura gerarchia impegnandosi a fare implementare i propri metodi "abstract" in modo da mentenere lo standard progettuale. | Da un punto di vista implementativo e funzionale, l'interfaccia stipula un contratto tra se e la classe o classe astratta che la eredita, obbligando queste ad implementare i metodi dell'interfaccia. |
| In poche parole, le classi non vi impongono nessun obbligo! | In poche parole, le classi astratte vi obbligano ad implementare i metodo "abstract" nella prima classe utile! | In poche parole, le interfacce vi obbligano a gestire i suoi metodi nelle classi o classi astratte. |

Cosa succede se una classe astratta eredita un interfaccia e quindi i suoi metodi?

Prima di rispondere a questa domanda vi ricordo i contratti che vengono stipulati:

a. L'interfaccia in questo caso stipula il contratto con la classe astratta che la eredita.
b. La classe astratta stipula a sua volta il contratto con l'intera struttura gerarchica.

Sulla base di questi contratti accade che:

a. I metodi dell'interfaccia vengono "affidati" sulla base del contratto alla classe astratta che se ne fa carico nel rispetto del contratto tra se e l'interfaccia.

b. Successivamente,la classe astratta deve anche rispettare il suo contratto con la struttura gerarchica, quindi i metodi ora in suo "affidamento",vengono "ri-affidati" alla struttura gerarchica, ossia alla prima classe figlia che avrà il compito di implementare tali metodi oltre a quelli eventuali "abstract" della stessa classe astratta.

Consigli per l'uso……

Esiste un prontuario che mi permette di scegliere in modo semplice cosa usare e quando?

La risposta è NO, perché in primis dovete studiare la situazione che il cliente vi richiede, le eventuali possibili modifiche future, l'evoluzione delle normative di legge che possono variare il software e molte altre cose….

La progettazione di una buona struttura gerarchica di classe è soggetta a tre pilastri:

a. Posso definire uno standard duraturo nel tempo?
   Se si allora posso pensare all'impiego di classi astratte.

b. Vi possono essere situazioni anormali o inconsuete per le quali non so cosa fare?
   Se si allora devo pensare all'impiego di interfacce.

c. Evitare di raggruppare tantissimi dati membro in una sola classe, ma applicare l'ereditarietà evitando così ridondanza dei dati membro.

ITI Serale
La prima scuola online

Domande finali!

a. Quando usare un'interfaccia?
b. Quando usare una classe astratta?
c. Se decido di usare una classe astratta questo significa che devo usare anche dei metodi "abstract"?
d. Quando uso una classe?
e. Tra classi,classi astratte ed interfacce quale strumento viene poi utilizzato in fase di run-time col linguaggio di programmazione?
f. Quali contratti esistono nella struttura gerarchica OOP?

Risposte!

Quando usare un’interfaccia?

Quando voglio descrivere dei casi particolari o rari per i quali non conosco nemmeno cosa accadrà ossia non conosco il codice da scrivere nei metodi dell’interfaccia, ecco perché vengono solo definiti ma non implementati, proprio per il fatto che varie situazioni non vi permettono di determinare a priori che istruzioni scrivere per gestire quella data situazione e verrete a conoscenza di tali informazioni solo ad un livello gerarchico più basso che vi permetterà di scrivere i suddetti metodi.

Risposte!

Quando usare una classe astratta?

Quando voglio definire uno standard di base dell'intera struttura gerarchica duraturo nel tempo.
Tale scelte va pensate bene sulla base delle esigenze del cliente e del problema che vi trovate a risolvere.

Risposte!

Se decido di usare una classe astratta questo significa che devo usare anche dei metodi “abstract”?

No, la classe astratta vi garantisce la definizione di uno standard,l’impiego e la successiva implementazione dei metodi astratti è necessaria quando questo standard è per sua natura sempre presente ma può assumere aspetti diversi nel tempo.
Un esempio potrebbe essere un software per la gestione delle paghe, il quale deve avere una standard duraturo di progettazione (classe astratta) ed uno strumento per il calcolo delle imposte capace di adeguarsi alle norme di legge che variano nel tempo.
Le imposte sono uno standard per tutti i lavoratori ecco quindi che devono essere gestite come metodo abstract e non con le interfacce

Risposte!

Quando uso una classe?

Quando voglio conglobare un insieme di dati membro (attributi) in modo da creare una struttura gerarchica che mi permetta poi alla fine di potere scrivere e leggere le informazioni all'interno di tale struttura sfruttando tutte le potenziali della OOP del linguaggio che sto utilizzando.

Risposte!

Tra classi,classi astratte ed interfacce quale strumento viene poi utilizzato in fase di run-time col linguaggio di programmazione?

Le classi astratte ed interfacce sono puri strumenti di progettazione gerarchica della OOP, anche la classe appartiene a questo gruppo ma in più ha il grande vantaggio di venire istanziata nello HEAP con l'operatore di allocazione dinamica "new" ottenendo così un oggetto.

Risposte!

Quali contratti esistono nella struttura gerarchica OOP?

Esistono solo due tipi di contratti.

1. Il primo può avere due contraenti diversi ossia:

   Tra interfaccia e classe.
   Tra interfaccia e classe astratta.

2. Tra classe astratta e struttura gerarchia ossia con la prima classe figlia che viene incontrata.