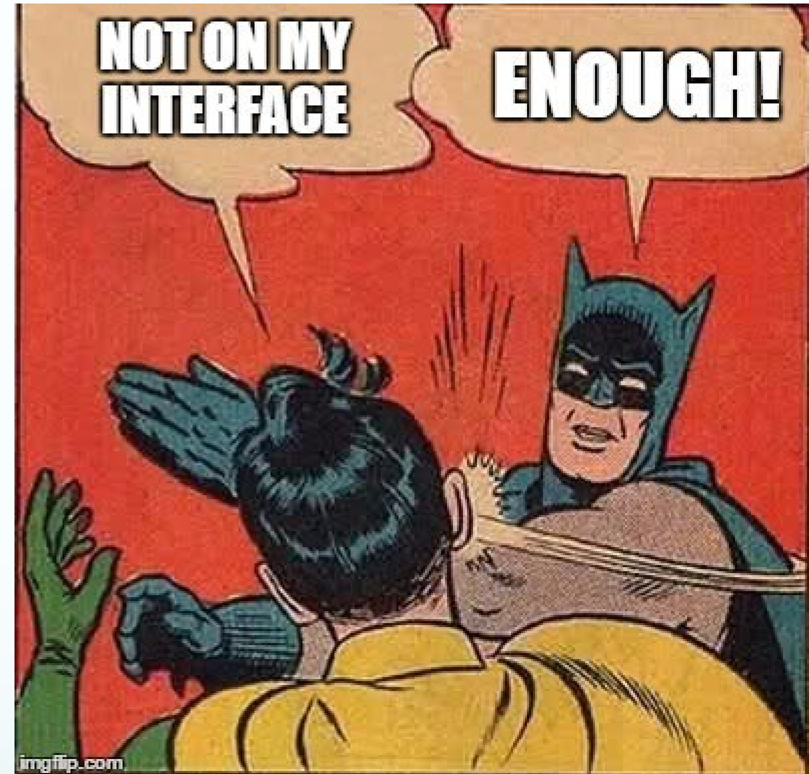
OOP:classi astratte e interfacce

Prof. Francesco Viglietti
www.in4matika.altervista.org



Classi astratte

In OOP, la realizzazione delle applicazioni inizia con la dichiarazione delle classi, elencando attributi e metodi, e prosegue con la creazione degli oggetti come istanze delle classi. In Java esistono delle classi particolari, ***classi astratte***, che non possono essere utilizzate per creare gli oggetti, esse si distinguono dalle altre classi perché hanno, di solito, almeno un metodo che non è stato implementato, ma è stato soltanto definito.
Una classe astratta definisce un'interfaccia senza implementarla completamente. Questo serve come base di partenza per generare una o più classi specializzate aventi tutte la stessa interfaccia di base, le quali potranno poi essere utilizzate indifferentemente da applicazioni che conoscono l'interfaccia base della classe astratta. La classe astratta da sola non può essere istanziata, viene progettata soltanto per svolgere la funzione di classe base da cui le classi derivate possono ereditare i metodi.

Classi astratte

public abstract class *NomeClasse {*
...
livelloDiVisibilità ***abstract*** *tipoRestituito nomeMetodo(parametri);*
}

Codice JAVA	Descrizione
`public abstract class Computer_classe_astratta`	Classe astratta
`{`	
`  public abstract void accendi();`	
`}`	
`public class PC extends Computer_classe_astratta`	Classe concreta
`{`	
`  public void accendi()`	Implementazione per il metodo accendi()
`  {`	
`      System.out.println("PC acceso");`	
`  }`	
`}`	

Esempio Figure geometriche

Si supponga di voler realizzare un'applicazione per calcolare l'area di diverse figure geometriche. Si può pensare di organizzare la gerarchia delle classi in modo da creare una super classe astratta Figura che imponga alle sue sottoclassi di avere un'implementazione per il metodo calcolaArea. Dichiarando le figure geometriche Quadrato e Cerchio come sottoclassi di Figura, si obbligano queste due sottoclassi a ridefinire il metodo calcolaArea al loro interno. La definizione della classe astratta Figura è la seguente:

Esempio F.G. codice classi

```
public abstract class Figura{
        public abstract double calcolaArea();
}
```

```
class Quadrato extends Figura{
     private double lato;
     public Quadrato(double lato){
         this.lato = lato;
     }
     public double calcolaArea() {
         return lato*lato;
     }
}
```

```
class Cerchio extends Figura{
     private double raggio;
     public Cerchio(double raggio) {
         this.raggio = raggio;
     }
     public double calcolaArea(){
         return raggio*raggio*Math.PI;
     }
}
```

Esempio F.G. main

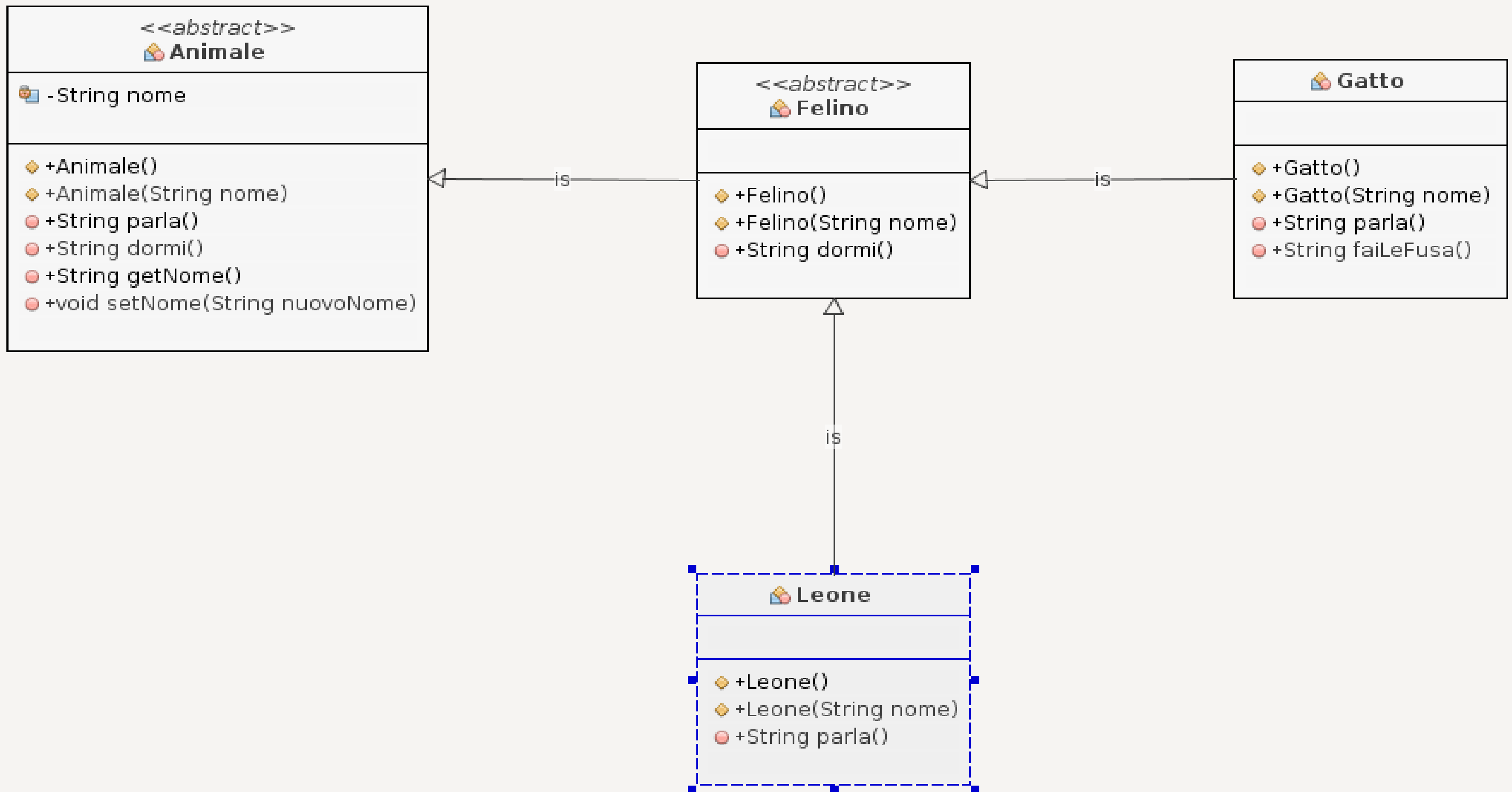
Il programma che calcola l’area di un quadrato e quella di un cerchio utilizza un oggetto di classe Figura a cui vengono successivamente assegnate le istanze della classe Quadrato e della classe Cerchio. Nel momento in cui viene richiesta l’esecuzione del metodo calcolaArea, il sistema di runtime di Java stabilisce qual è l’opportuno metodo da eseguire, a seconda che la figura sia un quadrato oppure un cerchio (polimorfismo). Il codice sorgente che calcola l’area di un quadrato e di un cerchio è il seguente:

```
class ProgFigura{
 public static void main(String args[]){
  Figura f;
  f = new Quadrato(4.5);
  System.out.println("Area = " + f.calcolaArea());
  f = new Cerchio(2.76);
  System.out.println("Area = " + f.calcolaArea());
  }
}
```

Esempio Animali

Supponiamo di avere la classe astratta Animale, che definisce i metodi astratti dorme e parla.
Da questa classe, deriva la classe Felino che è in grado di definire il metodo **dormi,** ma non è in grado di definire il metodo **parla**, perciò anche questa classe dovrà essere definita come astratta.
Da questa ultima classe vengono derivate le classi **Gatto** e **Leone** che sono in grado di definire il metodo **parla.**

Esempio animali UML



Esempio animali codice classi

```
public abstract class Animale {
private String nome;

public Animale(){}
public Animale(String nome){
	this.nome = nome;
}
public abstract String parla();
public abstract String dormi();
public String getNome() {
	return nome;
}
public void setNome(String nuovoNome) {
	this.nome = nuovoNome;
}
}
```

```
public abstract class Felino extends Animale {

public Felino() {}
public Felino(String nome) {
	super(nome);
}
public String dormi(){
	return "Ronf...";
}
}
```

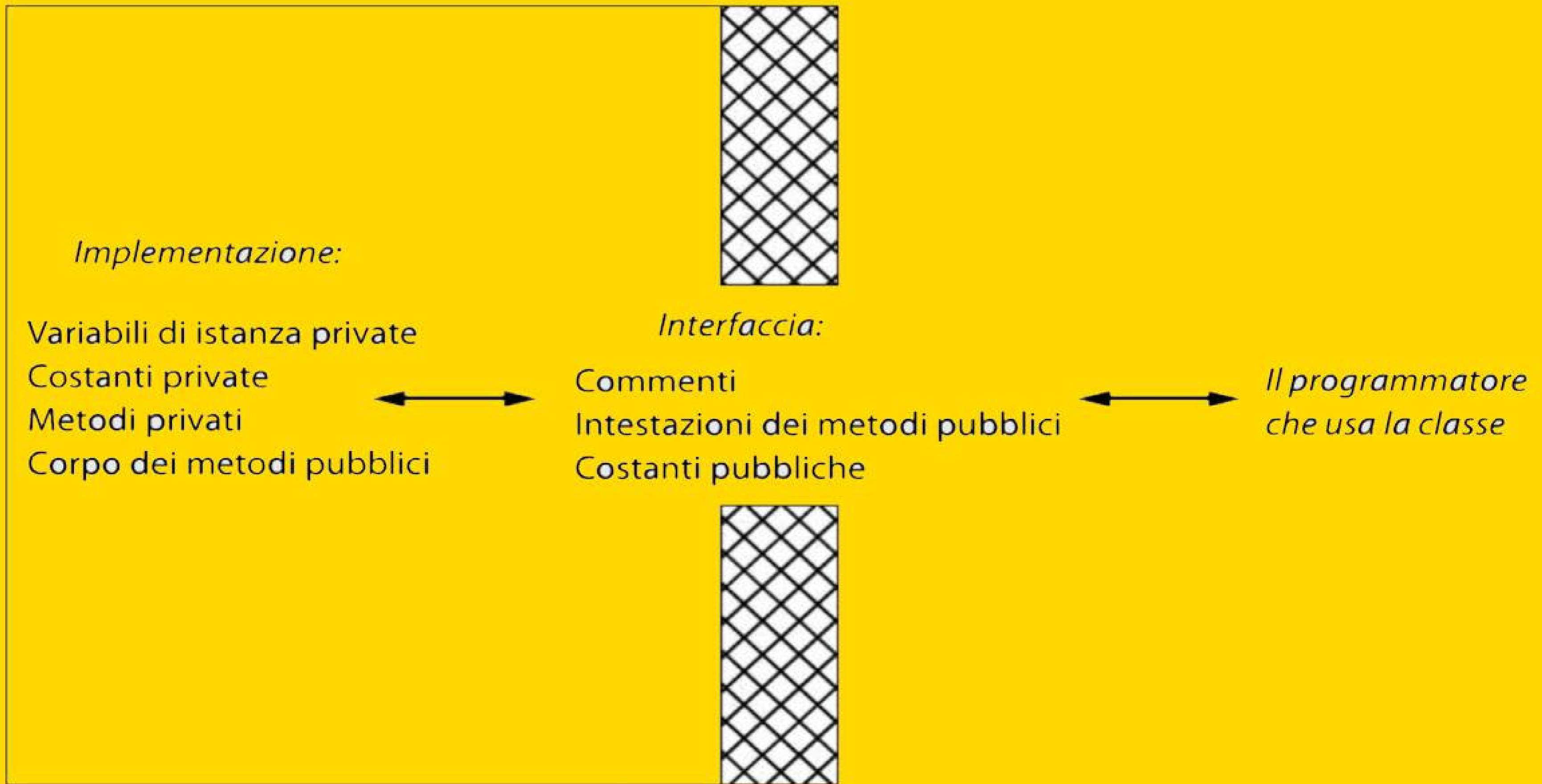
```
public class Gatto extends Felino {

public Gatto() {}
public Gatto(String nome) {
	super(nome);
}
public String parla() {
	return "Miao";
}
public String faiLeFusa(){
 return "prrrr";
}
```

```
public class Leone extends Felino {

public Leone() {}
public Leone(String nome) {
	super(nome);
}
public String parla() {
	return "Roarrrr";
}
```

Separazione classe



Interfacce java

Un'interfaccia è una sorta di classe astratta che dichiara dei metodi, cioè ne specifica solo l'intestazione, sarà compito delle classi che la implementano definirli. Per implementazione dell'interfaccia si intende la scrittura del codice dei metodi solo definiti da parte delle classi che "estenderanno" l'interfaccia. Pertanto, essa contiene, una serie di metodi astratti (implicitamente abstract) e può anche contenere degli attributi che devono essere inizializzati con un valore, poiché il compilatore li tratta automaticamente come final e static ovvero come costanti. Inoltre, sia i metodi astratti sia gli attributi costanti, sono implicitamente public.

Interfacce java

Da Java 8, le interfacce possono avere anche dei metodi con un'implementazione (il modificatore in questo caso sarà definito **default**) e tali metodi possono essere anche statici. In questo caso il metodo può essere implementato, nell'interfaccia stessa, e potrà essere modificato o meno nella classe che implementa l'interfaccia.

Un'interfaccia non può avere nessun costruttore.

Definizione:

[modifiers] interface MyInterface { }

Implementazione:

public class MyClass implements MyInterface {}

esempio

```
interface OnlyVar {
 public static final int NO = 0;
  public static final int YES = 1;
}
public class InterfacciaClient implements OnlyVar{
  public static void main(String args[]){
    System.out.println("NO = " + NO + " YES = " + YES);
  }
}
```

Sintassi

```
public interface nomeInterfaccia{
//definizioni costanti pubbliche
public static final tipo nomeCostante=valore
//definizioni metodi pubblici
public static tipoRestituito nomeMetodo (parametri);
```

implemetazione

Se una classe definisce tutti i metodi dell’interfaccia allora si dice che l’implementa. Se non li implementasse tutti la classe sarebbe astratta.
Una classe può anche implementare i metodi di più interfacce e può anche definire nuovi metodi non presenti nell’interfaccia.

Sintassi

```
public class nomeClasse implements interfaccia1,interfaccia2,...{
//corpo classe con implementazione dei metodi delle interfacce
}
```

Classi astratte vs interfacce

La scelta ricade tra una progettazione di un’ereditarietà per implementazione, nel caso di classi astratte, oppure di un’ereditarietà per interfaccia. Nel primo caso si definiscono (oltre alla dichiarazione di metodi astratti) anche dei metodi nella parte alta della gerarchia. Nel secondo caso si dichiarano nella parte alta della gerarchia solo metodi astratti, delegando alle altre classi, della parte bassa della gerarchia, la loro effettiva realizzazione (definizione). Le classi che implementano un’interfaccia si impegnano a definirne i metodi, se così non fosse, il compilatore genererà un messaggio di errore. Tuttavia, se la medesima classe non implementa i metodi dell’interfaccia, ma vuole delegare alle sue sottoclassi tale eventuale definizione, allora la stessa classe dovrà essere definita come abstract.

Tabella Classi astratte vs interfacce

	Interfacce	Classi astratte
Istanziabile	no	no
Fields	solo static final	sì
Costruttore	no	sì
Metodi statici	Da Java8	sì
Dichiarazione metodi	sì	sì
Implementazione metodi	Da Java8 (default)	sì

Usi in generale:

Quindi, cosa è meglio utilizzare? Ed in quali circostanze?

Si usa una classe astratta per condividere codice fra più classi, se più classi hanno in comune metodi e campi o se si vogliono dichiarare metodi comuni che non siano necessariamente campi static e final.

Si usa un’interfaccia se alcune classi, slegate fra di loro, si trovano a condividere i metodi di una interfaccia, se si vuole specificare il comportamento di un certo tipo di dato (ma non implementarne il comportamento) o se si vuole avere la possibilità di sfruttare la “multiple inheritance”.

Eredità nelle interfacce

Un'interfaccia può ereditare da altre interfacce (**extends**) ovviamente, la classe che la implementa dovrà fornire una definizione di tutti i metodi astratti della gerarchia di interfacce.
L'ereditarietà multipla si ha quando una classe può ereditare contemporaneamente da più superclassi, ovvero può avere più classi base. Una sorta di ereditarietà multipla si può tuttavia ottenere con le interfacce. Infatti, è consentito a una classe di implementare più interfacce.

public MyClass extends Bclass implements A, B, C

Come si vede la classe MyClass eredita dalla classe Bclass e implementa le interfacce A , B e C di cui dovrà definire tutti i metodi astratti.