# Programmazione Orientata agli Oggetti in Linguaggio Java

## Ereditarietà e Polimorfismo: Polimorfismo - b

### Interfacce e Classi Astratte

versione 1.2



**G. Mecca – Università della Basilicata – mecca@unibas.it**



## Sommario

- Il Concetto di Tipo
- Interfacce
- Classi Astratte
- Riassumendo

# Il Concetto di Tipo

- Nella programmazione procedurale
  - ⇨il tipo definisce le caratteristiche del dato
  - ⇨spazio di memoria, valori ammissibili e operazioni
- Nei linguaggi a oggetti
  - ⇨per gli oggetti la prospettiva è un po' diversa
  - ⇨si tratta infatti di componenti
  - ⇨che collaborano con altri componenti





# Il Concetto di Tipo

**ATTENZIONE**
al concetto di tipo come interfaccia

- Di conseguenza
  - ⇨c'è maggiore enfasi sui servizi offerti
  - ⇨e sul ruolo svolto nell'applicazione
- Tipo di un oggetto
  - ⇨corrisponde alla sua interfaccia
  - ⇨insieme di messaggi a cui l'oggetto è in grado di rispondere
  - ⇨due oggetti sono dello stesso tipo se sono in grado di rispondere agli stessi messaggi

# Il Concetto di Tipo

- Una classe
  - ⇨definisce un tipo (interfaccia per gli oggetti)
  - ⇨e contemporaneamente fornisce anche un'implementazione di quel tipo (codice)
- Ma, come abbiamo visto...
  - ⇨in alcuni casi è necessario definire il tipo degli oggetti da manipolare
  - ⇨senza specificarne esplicitamente l'implementazione





# Il Concetto di Tipo

- I linguaggi di programmazione OO
  - ⇨forniscono uno strumento per definire tipi, senza associargli implementazioni
  - ⇨le "interface"
  - ⇨e uno strumento intermedio tra le classi e le interface: le classi astratte

# Interfacce

- Interface
  - ⇨collezione di prototipi di metodi
  - ⇨definisce puramente un tipo nell'applicazione
  - ⇨lasciando poi alle classi l'implementazione
- Sintassi

```
public interface <nome> {
  <prototipi dei metodi>
}
```





```
package it.unibas.volpieconigli.modello;

public interface Animale {

   void agisci (Scacchiera scacchiera);

   boolean morto ();

   String toString();

   public boolean isConiglio();

}
```

```
package it.unibas.appuntamenti.modello;

public interface Impegno extends Comparable {

    public void setGiorno(Giorno giorno);

    public Giorno getGiorno();

    public Orario getOrario();

    public java.lang.String getLuogo();

    public void setLuogo(java.lang.String luogo);

    public java.lang.String getNote();

    public void setNote(java.lang.String note);

    public abstract String toString();

    public abstract String saveString();
}
```

Impegno estende l'interfaccia Comparable

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





# Interfacce

- Note sulla sintassi
  - ⇨i prototipi dei metodi sono considerati automaticamente pubblici, anche se non viene specificato public
  - ⇨un'interfaccia può estenderne altre; in questo caso i prototipi della superinterfaccia fanno parte anche della sottointerfaccia
  - ⇨un'interfaccia può contenere proprietà, che sono però automaticamente considerate public, final e static (costanti)

# Interfacce

- A cosa serve un'interfaccia ?
  - ⇨a definire un "contratto"
  - ⇨ovvero una serie di impegni per le classi i cui oggetti la implementeranno
  - ⇨le classi si impegnano a fabbricare oggetti che rispettino il tipo definito dall'interfaccia
  - ⇨quindi un'interfaccia serve a poco senza delle classi che la implementino





# Interfacce

- Sintatticamente
  - ⇨per dichiarare di implementare un'interfaccia, una classe deve usare la parola implements
  - ⇨questo impone alla classe di implementare tutti i metodi dell'interfaccia
- Esempi
  - ⇨public class Volpe implements Animale {...}
  - ⇨public class Coniglio implements Animale {...}

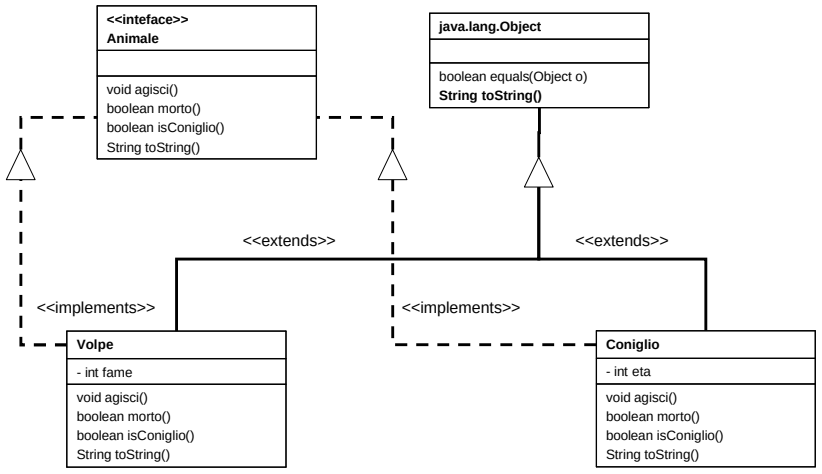
# Interfacce

- Semantica di implements
  - ⇨la classe si impegna ad implementare tutti i metodi previsti dall'interfaccia
  - ⇨la classe "eredita" il tipo dell'interfaccia
  - ⇨ovvero: gli oggetti della classe sono utilizzabili dovunque sia richiesto un oggetto del tipo dell'interfaccia
  - ⇨infatti: sono in grado di ricoprire quel ruolo





# Un Ulteriore Esempio



# Interfacce

- Attenzione
  - ⇨mentre c'è il vincolo di ereditarietà singola per l'estensione, questo non vale per l'implementazione di interfacce
  - ⇨una classe può implementare interfacce diverse
- Esempio

```
public class Volpe implements Animale, Serializable {
   ...
}
```





# Interfacce

- Di conseguenza
  - ⇨esistono due forme diverse di unità di scrittura del codice
  - ⇨la classe, che definisce un tipo e ne fornisce contestualmente un'implementazione
  - ⇨l'interfaccia, che definisce puramente un tipo senza implementazione
  - ⇨in mezzo c'è un ulteriore categoria: la classe astratta

# Classi Astratte

- Classe astratta
  - ⇨classe a metà tra la classe ordinaria (o "classe concreta") e l'interfaccia
  - ⇨fornisce una implementazione parziale di un'interfaccia
  - ⇨lasciando alcuni metodi dell'interfaccia in forma astratta, ovvero non implementata
  - ⇨deve essere estesa da una classe concreta che implementa i metodi astratti





# Classi Astratte

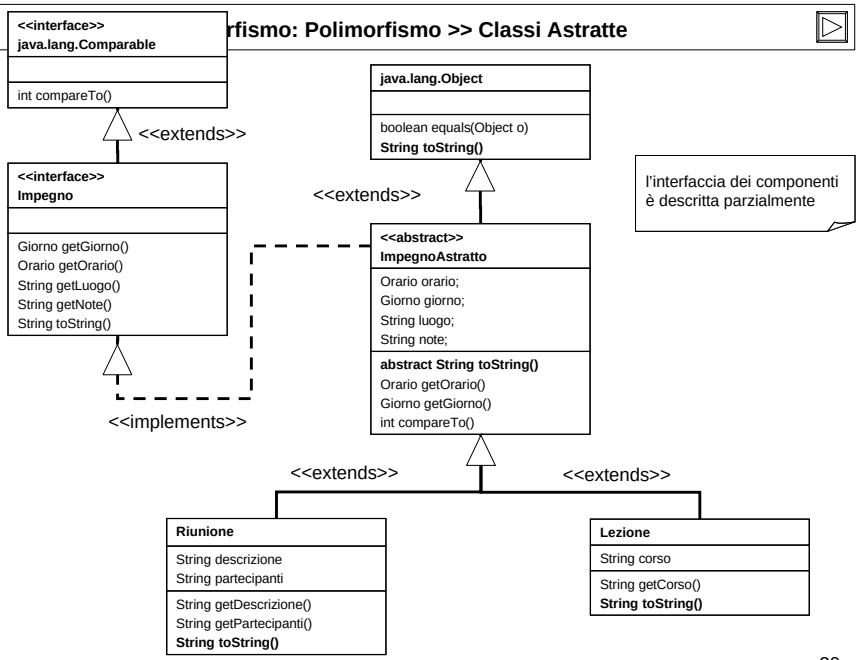
- In altri termini
  - ⇨una classe astratta ha senso solo come padre in una gerarchia
  - ⇨serve a mettere codice a "fattor comune"
  - ⇨devono esserci classi concrete che la estendono implementando i metodi astratti
- Sintatticamente
  - ⇨le classi astratte e i metodi astratti sono indicati con la parola chiave abstract

## Ereditarietà e Polimorfismo: Polimorfismo >> Classi Astratte

```
package it.unibas.appuntamenti.modello;

public abstract class ImpegnoAstratto implements Impegno {

   protected String luogo,  note;
   protected Orario orario;
   protected Giorno giorno;

   protected ImpegnoAstratto(String luogo, String note, Orario orario) {
      this.luogo = luogo;     this.note = note;  this.orario = orario;
   }

   public java.lang.String getLuogo() {
      return luogo;
   }

   public void setLuogo(java.lang.String luogo) {
      this.luogo = luogo;
   }
   ...

   public abstract String toString();

   public abstract String saveString();

}
```



## rfismo: Polimorfismo >> Classi Astratte



# Classi Astratte

- Attenzione
  - ⇨avremmo potuto utilizzare anche una classe concreta invece che astratta
  - ⇨definendo il metodo toString() in qualche modo
- Esempio: in ImpegnoAstratto

```
public String toString() {
    return this.orario + " – " + this.luogo + " " + this.note;
}
```





# Classi Astratte

- In Lezione (e similmente in Riunione)

```
public String toString() {
    return super.toString() + " – " + this.corso;
}
```

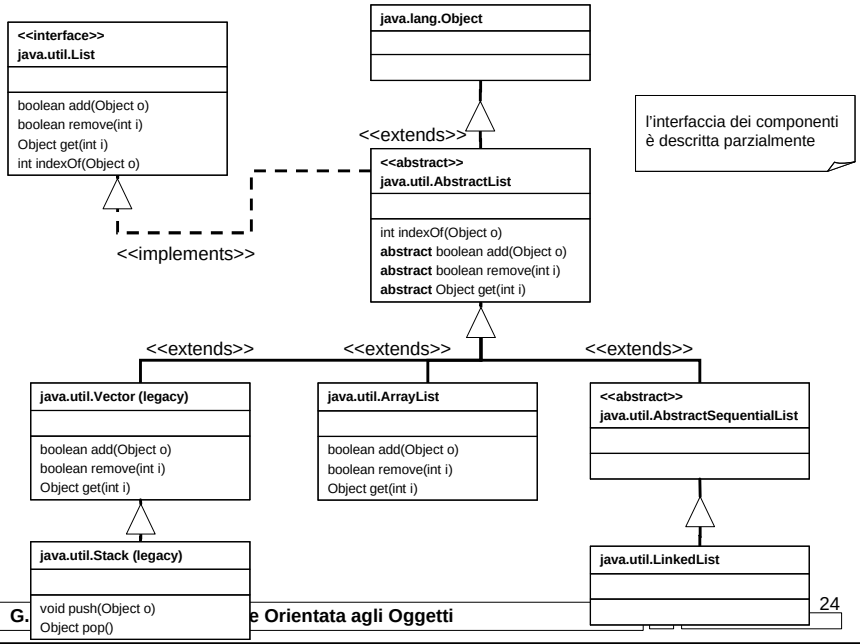
- In questo modo
  - ⇨la gerarchia sarebbe costituita esclusivamente da classi concrete oltre all'interfaccia

# Classi Astratte

- Ma..
  - ⇨in questo caso la soluzione è un pò forzata (non ha molto senso il metodo toString() di ImpegnoAstratto)
- Inoltre
  - ⇨è tipico il caso di una gerarchia composta di un'interfaccia, una classe astratta e varie classi concrete
  - ⇨es: le liste








# Classi Astratte

- Un vincolo sulle classi astratte
  - ⇨non è possibile creare direttamente oggetti di queste classi
- Spiegazione
  - ⇨questi oggetti avrebbero un'implementazione incompleta e non sarebbero capaci di eseguire tutti i metodi relativi
  - ⇨di conseguenza, gli oggetti della classe astratta sono creati solo a seguito di creazione di oggetti nelle loro sottoclassi





# Riassumendo

- Il concetto di tipo nella POO
  - ⇨tipo = interfaccia
  - ⇨insieme dei servizi che un componente può fornire nell'applicazione
- Vari strumenti per definire tipi
  - ⇨classi
  - ⇨interface
  - ⇨classi astratte

# Riassumendo

- Una classe (concreta)
  - ⇨definisce un'interfaccia (un tipo) per i propri oggetti
  - ⇨e ne specifica contestualmente anche l'implementazione
  - ⇨es: la classe Volpe definisce il tipo Volpe e lo implementa
  - ⇨es: la classe Coniglio definisce il tipo Coniglio e lo implementa





# Riassumendo

- Una interface
  - ⇨definisce puramente un tipo (insieme di servizi)
  - ⇨senza specificarne minimamente l'implementazione; la definizione dell'implementazione è lasciata alle classi che implementano l'interfaccia
  - ⇨es: Animale definisce un tipo animale implementabile in vari modi

# Riassumendo

- Una classe astratta
  - ⇨definisce un tipo
  - ⇨e ne fornisce anche un'implementazione parziale
  - ⇨lasciando i dettagli rimanenti alle classi che estendono la classe astratta





# Riassumendo

- Di conseguenza
  - ⇨abbiamo risolto il problema n. 1
  - ⇨come definire un tipo comune alle varie implementazioni
  - ⇨basta definire un'interface
- Attenzione
  - ⇨in alcuni casi l'interface non sarebbe strettamente necessaria

# Riassumendo

- Esempio n. 1: volpi e conigli
  - ⇨l'interface Animale è indispensabile
  - ⇨non esiste un tipo comune che descriva sia Volpe che Coniglio
- Esempio n. 2: appuntamenti (o liste)
  - ⇨in questo caso, tecnicamente, basterebbe la classe astratta (ImpegnoAstratto)





# Riassumendo

```
public static void stampa (ArrayList lista) {      // con l'interfaccia
   for (int i = 0; i < lista.size(); i++) {
      Impegno impegno = (Impegno)lista.get(i);
      System.out.println(impegno.toString());
   }
}

public static void stampa (ArrayList lista) {      // senza l'interfaccia
   for (int i = 0; i < lista.size(); i++) {
      ImpegnoAstratto impegno = (ImpegnoAstratto)lista.get(i);
      System.out.println(impegno.toString());
   }
}
```

# Riassumendo

- Ma...
  - ⇨una classe astratta contiene dei dettagli implementativi (anche se parziali)
  - ⇨i dettagli implementativi possono cambiare nel tempo
  - ⇨il tipo invece non dovrebbe cambiare
- Consuetudine nella POO
  - ⇨definire i tipi attraverso le interface anche in presenza di classi astratte





# Riassumendo

- Il Concetto di Tipo
- Interfacce
- Classi Astratte
- Riassumendo

# Termini della Licenza