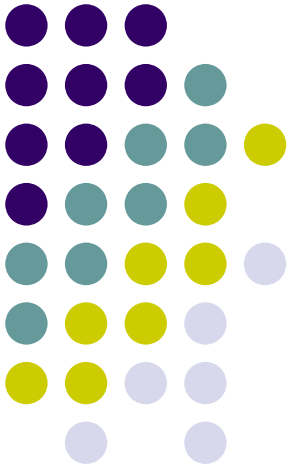
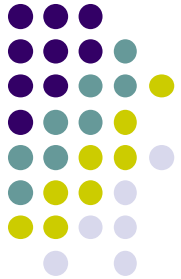
orso di Programmazione

Java Interface

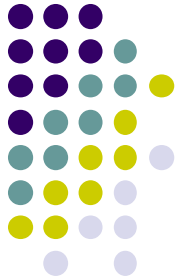


Gli oggetti software comunicano tramite interfacce



- Gli oggetti software comunicano tramite interfacce.
- Un'interfaccia Java descrive una serie di metodi che possono essere invocati su un particolare oggetto per dirgli di svolgere un compito o restituire qualche informazione.

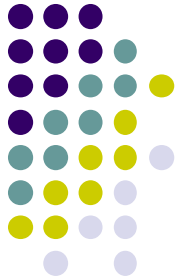
Interfacce in Java



- Una dichiarazione di interfaccia inizia con la parola chiave *interface*.
- Quando il meccanismo fu introdotto un'interfaccia poteva contenere solo costanti e metodi astratti.
- Tutti i membri di un'interfaccia devono essere *public* e l'interfaccia non può specificare alcuni dettagli implementativi:
  - *non sono consentite dichiarazioni di variabili di istanza;*
    - *gli attributi sono implicitamente public, static e final*
  - *In generale i metodi dichiarati in un'interfaccia sono implicitamente metodi pubblici astratti;*

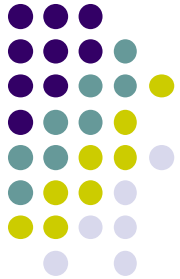
- Un'interfaccia deve essere dichiarata in un file che ha il suo stesso nome ed estensione .java.

Quando usare una interfaccia 1/3



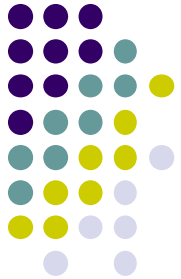
- Spesso si utilizza un'interfaccia quando classi non imparentate (ovvero classi non collegate da una gerarchia di classe) devono condividere alcuni metodi (comportamenti) e costanti comuni.
  - Questo consente a istanze di classi non imparentate di essere elaborate in maniera polimorfica; gli oggetti appartenenti a classi che implementano la stessa interfaccia rispondono alle stesse invocazioni (per i metodi di quella interfaccia).
- Si può creare un'interfaccia che descrive la funzionalità richiesta e implementarla in tutte le classi che richiedono quella funzionalità.

Quando usare una interfaccia 2/3



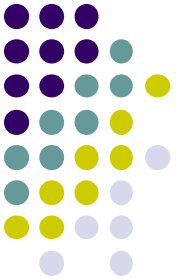
- *Implementare un'interfaccia è come firmare un contratto con l'utente che dice: "dichiarerò tutti i metodi specificati dall'interfaccia oppure dichiarerò la mia classe come astratta".*
- Contratto *per separare nettamente l'interfaccia dalle sue possibili implementazioni.*
  - *Per la realizzazione di ADT.*

Quando usare una interfaccia 3/3



- Per gestire la situazione comune di una classe che deve riflettere il comportamento di due o più classi padre
  - Meccanismo denominato *ereditarietà multipla.*
    - ***Non ammessa in Java ma ammessa in C++.***

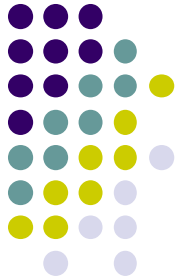
Classi Astratte o Interfacce?



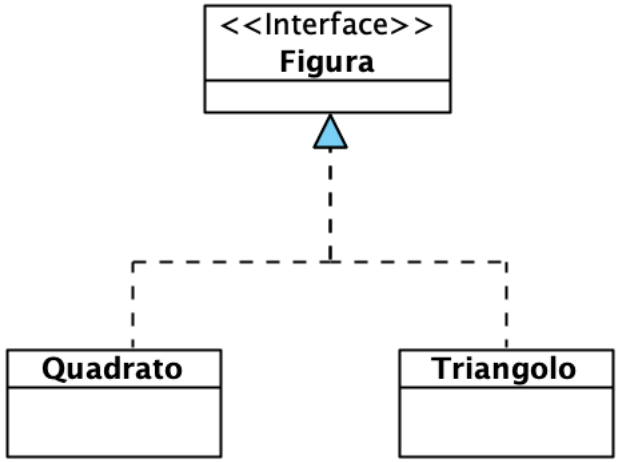
- Molti sviluppatori considerano le interfacce una tecnologia di modellazione persino più importante delle classi astratte.

- Un'interfaccia dovrebbe essere usata al posto di una classe astratta quando non ci sono implementazioni (o molto ridotte) predefinite da ereditare, cioè nessun campo e nessuna implementazione di metodo.

Realizzazione di una interfaccia

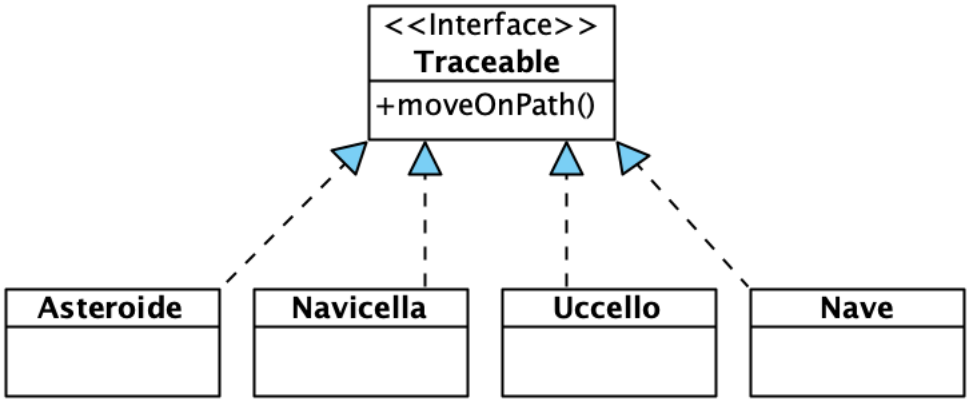


- Tutti gli oggetti di una classe che **implementano** interfacce multiple hanno una relazione *è-un* con ogni interfaccia implementata.
- Per usare un'interfaccia, una classe concreta deve specificare il fatto che la implementa (con la parola chiave **`implements`**) e deve definire ogni metodo con il medesimo prototipo indicato nell'interfaccia.

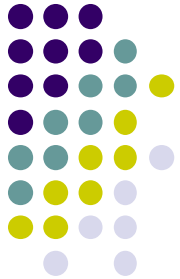


Esempio – caso1

- Cosa possono avere in comune i pianeti, gli asteroidi, le navicelle spaziali, gli uccelli migratori e le navi?
- Si muovono ad esempio lungo un percorso/traiettoria (abbastanza) prestabilito.
  - Condividono tutte lo stesso comportamento quindi potrebbero implementare la stessa interfaccia



Esempio – caso2

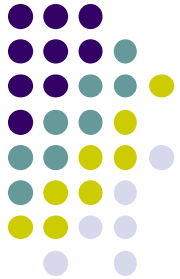


```
public interface Animale {
    public abstract double move();
    public abstract void verso();
}
```

```
public class Leopardo implements Animale {

    @Override
    public void verso() {
        System.out.println(».. leopardo");
    }

    @Override
    public double move() {
        return 1.0;
    }
}
```

```
public class Tartaruga implements Animale {

    @Override
    public void verso(){
        System.out.println(".. tartaruga");
    }

    @Override
    public double move(){
        return 0.1;
    }
}
```

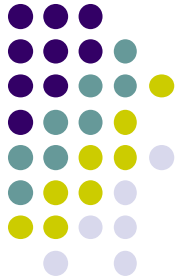
Esempio – caso 2



```
public static void main(String[] args) {
        Leopardo l1 = new Leopardo();
        Leopardo l2 = new Leopardo();
        Tartaruga t1 = new Tartaruga();

        l1.verso();
        System.out.println("Leopardo l2 si muove di "+l2.move());
        t1.verso();
        System.out.println("La tartaruga si muove di "+l2.move());
}
```

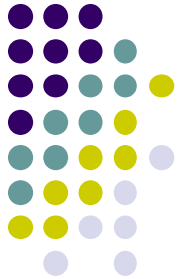
```
run:
.. leopardo
Leopardo l2 si muove di 1.0
.. tartaruga
La tartaruga si muove di 1.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Errori tipici



- *Se in una classe concreta che implementa un'interfaccia (con la parola chiave* implements*) non si implementano tutti i metodi* abstract *dell'interfaccia, si determina un errore di compilazione che indica che la classe deve essere dichiarata* abstract*.*

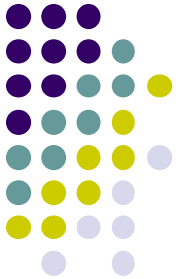
Interfacce e polimorfismo



- L'interfaccia Animale può essere usata per elaborare un insieme di Leopardo e Tartaruga in maniera polimorfica

```
public static void main(String[] args) {
        Animale[] zoo = new Animale[4];
        zoo[0]= new Leopardo();
        zoo[1]= new Tartaruga();
        zoo[2]= new Tartaruga();
        zoo[3]= new Leopardo();
        for (Animale animale : zoo) {
            animale.verso();
            System.out.println("Mi muovo di "+animale.move());
        }
    }
```

Interfacce e polimorfismo



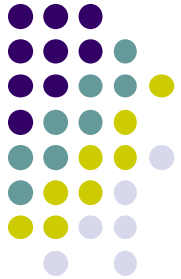
- L'interfaccia Animale può essere usata per elaborare un insieme di Leopardo e Tartaruga in maniera polimorfica

```
public static void main(String[] args) {
        Animale[] zoo = new Animale[4];
        zoo[0]= new Leopardo();
        zoo[1]= new Tartaruga();
```

```
run:
.. leopardo
Mi muovo di 1.0
.. tartaruga
Mi muovo di 0.1
.. tartaruga
Mi muovo di 0.1
.. leopardo
Mi muovo di 1.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
imale.move());
```

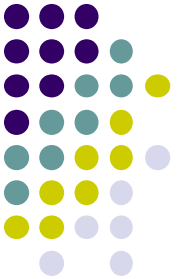
Interfacce per la risoluzione di ereditarietà multipla



- Java non consente alle sottoclassi di ereditare da più di una singola superclasse, ma permette a una classe di ereditare da una superclasse e implementare tutte le interfacce necessarie.
- Per implementare più di una interfaccia, basta inserire un elenco di nomi di interfaccia separati da una virgola dopo la parola chiave implements nella dichiarazione della classe, come in:

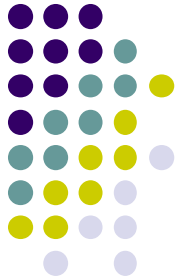
```
public class NomeClasse implements PrimaInterfaccia,
SecondaInterfaccia, ...
```

Ereditarietà tra interfacce

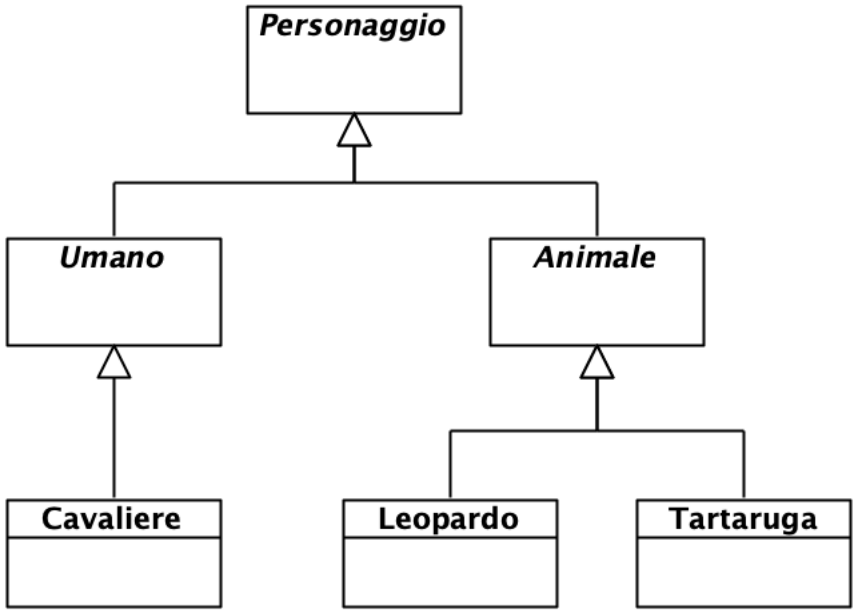


- Una interface può ereditare (`extends`) da un'altra interface.
- Una classe astratta può realizzare (`implements`) una o più interface.

Esempio – caso 3

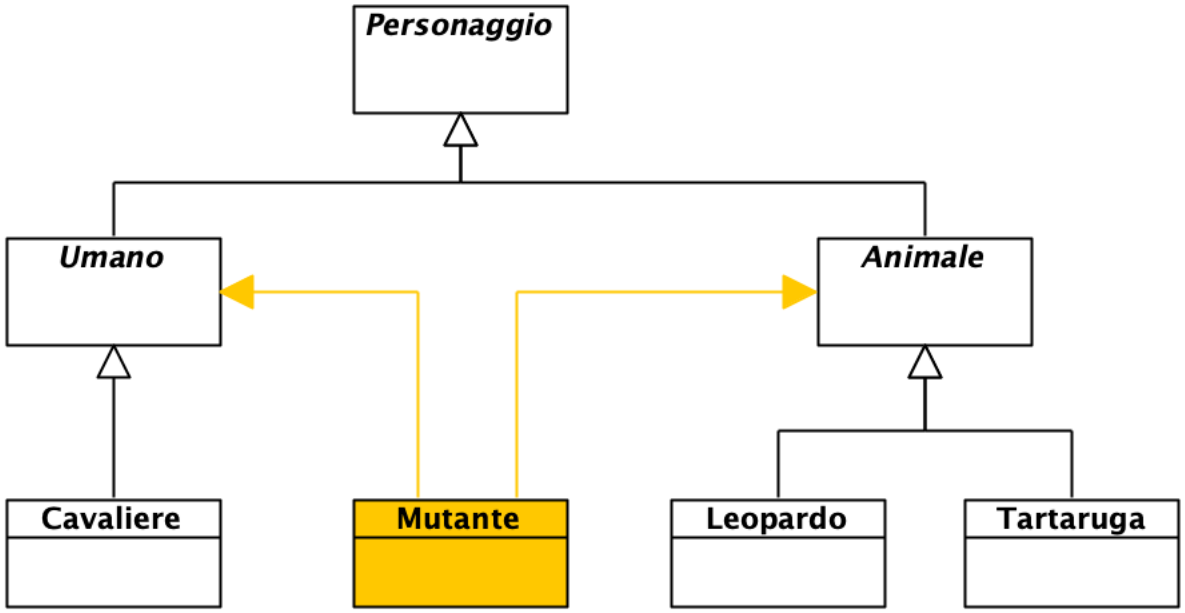


- Supponiamo di voler implementare un gioco i cui personaggi sono umani e animali.
- Possiamo creare due classi astratte `Umano` e `Animale` da cui ereditano specifici personaggi.



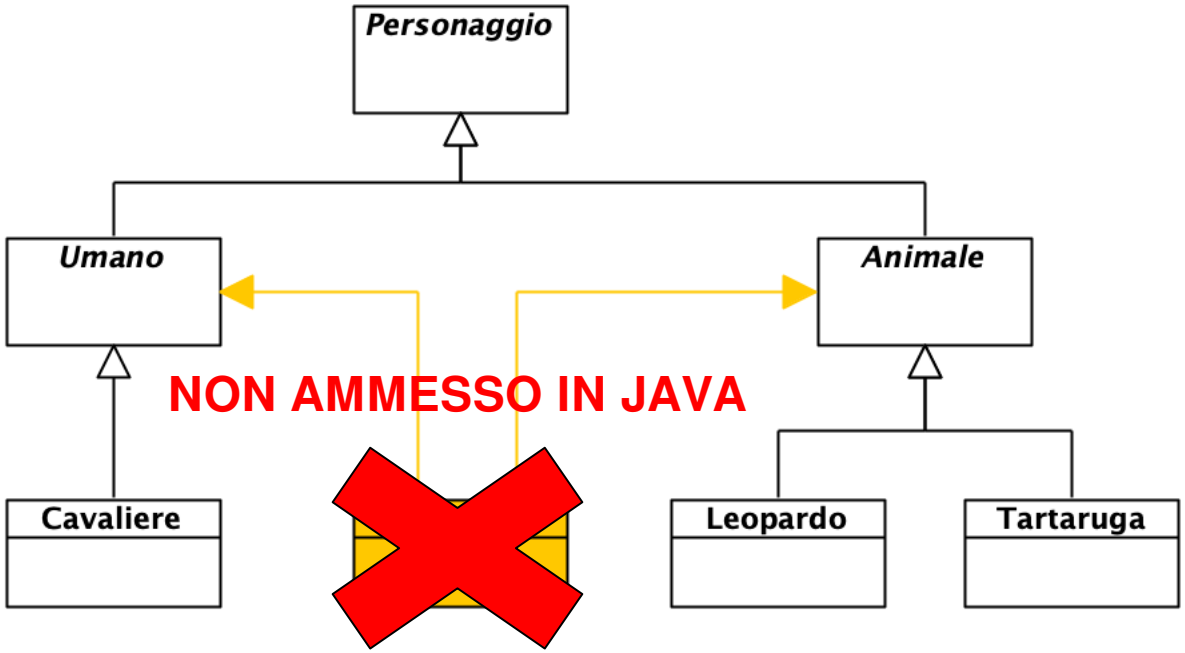
Esempio – caso 3

- Ma se volessimo avere un altro tipo di personaggio denominato Mutante?
  - Avente le caratteristiche di Umano e Animale
  - Di fatto la classe dovrebbe ereditare sia da Umano che da Animale



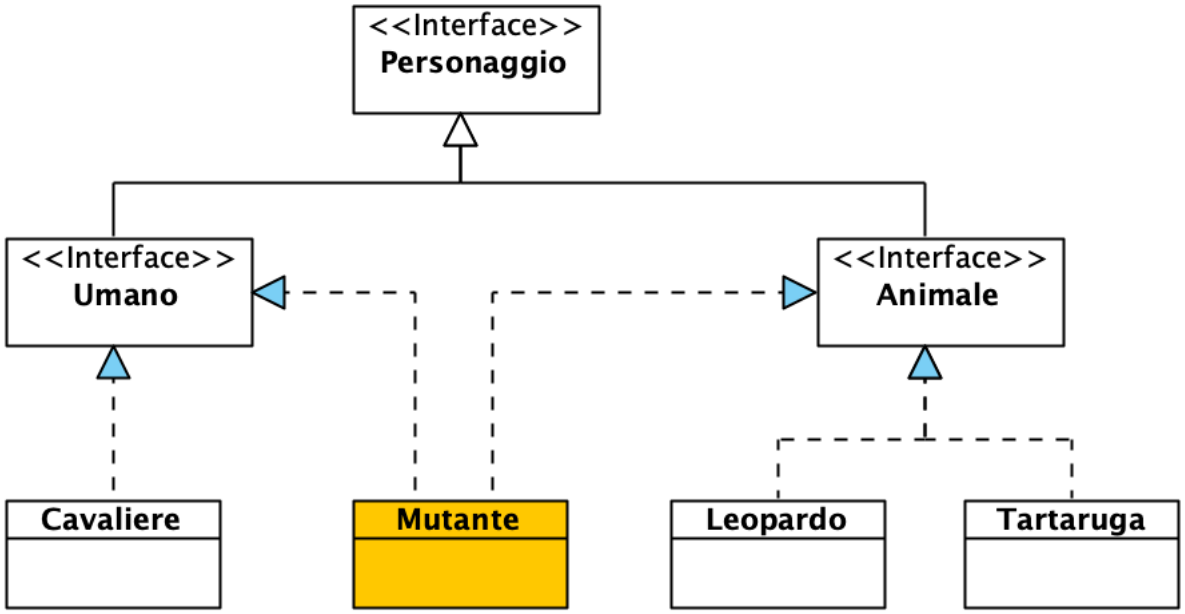
Esempio – caso 3

- Ma se volessimo avere un altro tipo di personaggio denominato Mutante?
  - Avente le caratteristiche di Umano e Animale
  - Di fatto la classe dovrebbe ereditare sia da Umano che da Animale

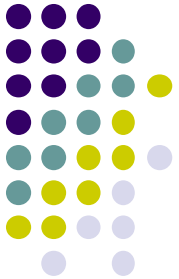


Soluzione – caso3

- Potremmo considerare Personaggio, Umano e Animale come Interfacce.
  - In questo modo la classe Mutante può realizzare (implementare) sia l'interfaccia Umano che l'Interfaccia Animale

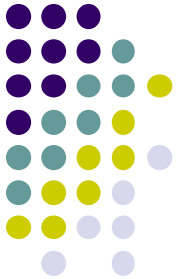


In Java



```
public interface Personaggio {
    public abstract double muovi();
    public abstract double colpisci();
}
```

In Java



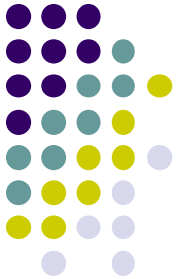
```
public interface Animale extends Personaggio{
    //sono di default static final
    //da inizializzare
    public int DANNO_BASE_ANIMALE=10;
    public int MOVIMENTO_BASE_ANIMALE=5;

    public abstract void verso();}
```

In Java

```
public interface Umano extends Personaggio{
    //sono di default static final
    //da inizializzare
    public int DANNO_BASE_UMANO=5;
    public int MOVIMENTO_BASE_UMANO=15;

    public abstract void parla();
}
```

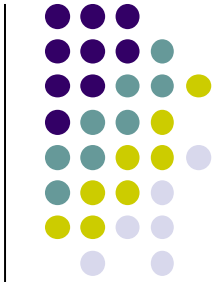
- Se le variabili avessero avuto lo stesso nome nella classe Mutante, mostrata nella slide successiva ci sarebbe stato un problema di inconsistenza ✉ Errore di compilazione.
- In generale il problema si avrebbe per tutte le classi che implementerebbero entrambe le interfacce

In Java



```
public class Mutante implements Umano, Animale{
    @Override
    public void verso(){
        System.out.println("..mutante");
    }

    @Override
    public double muovi(){
        return 15*MOVIMENTO_BASE_UMANO;
    }

    @Override
    public double colpisci(){
        return 18*DANNO_BASE_ANIMALE;
    }

    @Override
    public void parla(){
        System.out.println("Sono un mutante");
    }

}
```

In Java



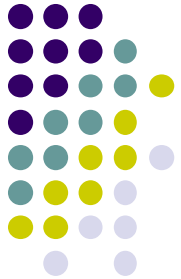
```
public static void main(String[] args) {
        //verifichiamo l'uso della variabile static
pubblica
        System.out.println("Il danno base di un
personaggio Animale è: " + Animale.DANNO_BASE_ANIMALE);

        Animale[] branco = new Animale[4];
        branco[0]= new Leopardo();
        branco[1]= new Tartaruga();
        branco[2]= new Mutante(); //Ammesso perché anche
Animale
        branco[3]= new Leopardo();

        for (Animale animale : branco) {
            animale.verso();
            System.out.println("Mi muovo di
"+animale.muovi());
        }
    }
```

In Java

```
public static void main(String[] args) {
        //verifichiamo l'uso della variabile static
pubblica
        System.out.println("Il danno base di un
personaggio Animale è: " + Animale.DANNO_BASE_ANIMALE);

        Animale[] branco = new Animale[4];
        branco[0]= new Leopardo();
        branco[1]= new Tartaruga();
```

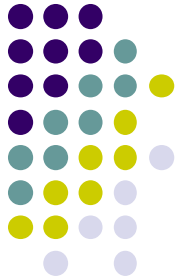
```
run:
Il danno base di un personaggio Animale è: 10
.. leopardo
Mi muovo di 1.0
.. tartaruga
Mi muovo di 0.1
..mutante
Mi muovo di 225.0
.. leopardo
Mi muovo di 1.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Miglioramenti in Java SE 8



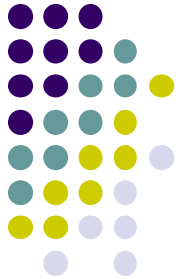
- A partire da *Java SE 8* sono state aggiunte nuove funzionalità di interfaccia che le rendono ancora più flessibili.

Miglioramenti: metodi di default



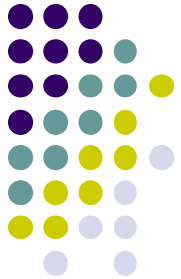
- Prima di Java SE 8, i metodi di interfaccia potevano essere solo public abstract.
  - un'interfaccia specificava quali operazioni doveva eseguire la classe dedicata all'implementazione ma non come farlo.
- A partire da Java SE 8, le interfacce possono anche includere metodi default di tipo public con implementazioni di default concrete che specificano come vengono eseguite le operazioni quando la classe che implementa l'interfaccia non ridefinisce i metodi.
- Se una classe implementa un'interfaccia, ne riceve anche le implementazioni default.
- Un metodo default, è dichiarato tramite la parola chiave default prima del tipo di ritorno del metodo
  - Bisogna fornire un'implementazione di metodo concreta.
- È ammesso l'override dei metodi di default delle classi concrete che realizzano le interfacce.

Confronto tra interfacce e classi abstract



- Prima di Java SE 8 si utilizzava solitamente un'interfaccia (piuttosto che una classe abstract) quando non c'erano dettagli di implementazione da ereditare
  - né campi né implementazioni di metodo.
- Con i metodi default, si possono invece dichiarare comuni implementazioni di metodo nelle interfacce.
- Questo consente una maggiore flessibilità nella progettazione delle classi, perché una classe può implementare molte interfacce ma estendere una sola superclasse.

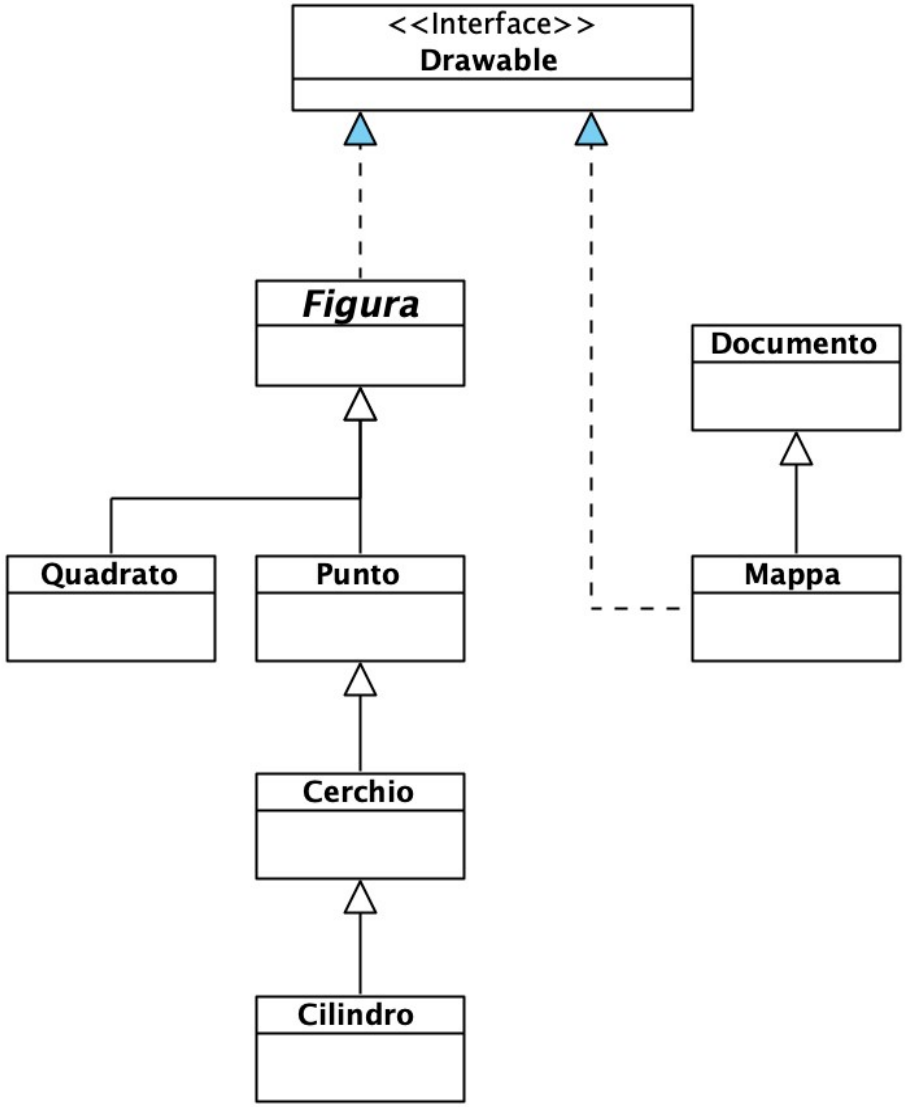
Esempio

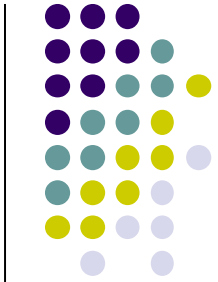


```
public interface Animale {
    //sono di default static final
    //da inizializzare
    public int DANNO_BASE_ANIMALE=10;
    public int MOVIMENTO_BASE_ANIMALE=5;

    public abstract double muovi();
    public abstract double colpisci();
    public abstract void verso();

    public default double corri(){
        return 100*MOVIMENTO_BASE_ANIMALE;
    }
}
```

Esempio finale

- Sviluppiamo un esempio che comprende ereditarietà di classi, classi astratte e intefacce
- Esempio delle figure rappresentato dal class diagram.





Riferimenti

- Programmare in Java:
  Capitolo 10, §10.9 e §10.10