UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Università di degli Studi di Salerno
Dipartimento di Informatica

Programmazione ad Oggetti

a.a. 2023-2024

Interfacce

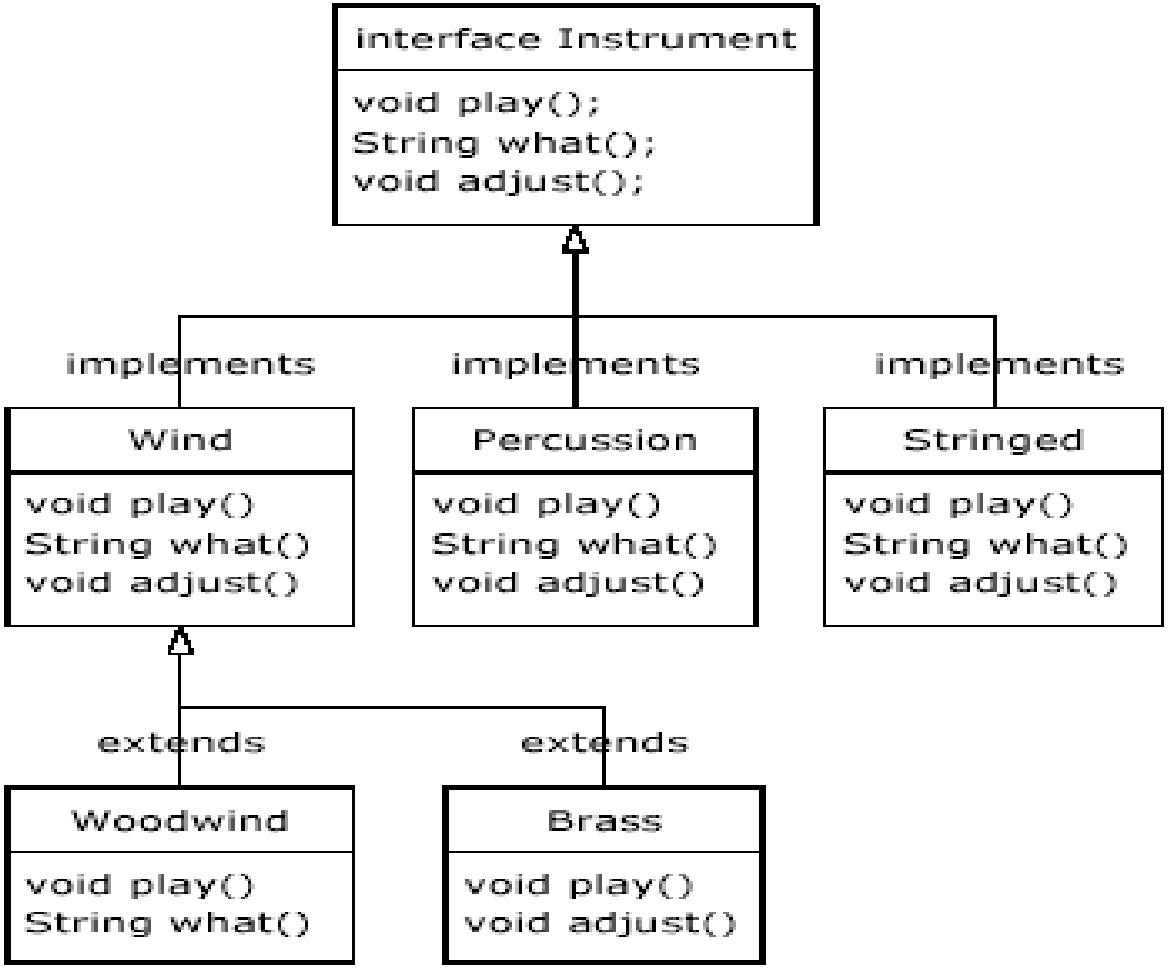
Docente: Prof. Massimo Ficco

E-mail: *mficco@unisa.it*

Interfacce

- Le interfacce definiscono classi completamente astratte
- Ogni metodo dichiarato è di default pubblico e non può avere implementazione
- Hanno solo attributi ***statici, final*** ed **inizializzati** (*no blank static*)
- I metodi di un'interfaccia hanno automaticamente visibilità public, e la classe che li implementa deve dichiarare una visibilità public, altrimenti per default sono "friendly".

Esempio


interface Instrument
void play();
String what();
void adjust();
implements
implements
implements
Wind
void play()
String what()
void adjust()
Percussion
void play()
String what()
void adjust()
Stringed
void play()
String what()
void adjust()
extends
extends
Woodwind
void play()
String what()
Brass
void play()
void adjust()

Esempio

```
interface Instrument {
  // Compile-time constant:
  int i = 5;                          // static & final
  // Cannot have method definitions:
  void play(Note n);                  // Automatically public
  String what();
  void adjust();
}

class Wind implements Instrument {
        public void play(Note n) {System.out.println("Wind.play() " + n);}
        public String what() { return "Wind"; }
        public void adjust() {}
}

class Woodwind extends Wind {
        public void play(Note n) {System.out.println("Woodwind.play() " + n);}
        public String what() { return "Woodwind"; }
}
```

Ereditarietà multipla

Il c++ supporta l'ereditarietà multipla: ereditare più classi

L'ereditarietà multipla può causare problemi di collisione

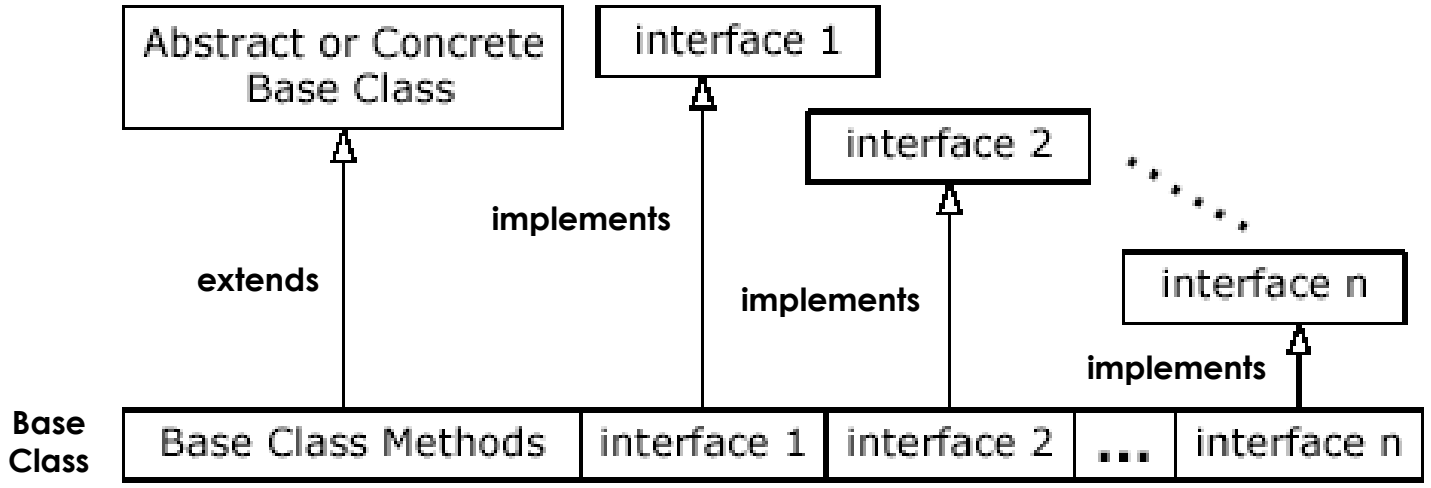
Occorre supportare l'upcasting verso diverse classi

Le ***interfacce*** permettono di gestire in modo più sofisticato gli oggetti di un progetto

Una classe può implementare più interfacce attraverso la parola chiave ***implements***

Ereditarietà multipla

In Java una classe può ereditare una sola classe e implementare più interfacce.



Esempio

```
interface CanFight {
   void fight();
}
interface CanSwim {
   void swim();
}
interface CanFly {
   void fly();
}
class ActionCharacter {
    public void actionfight() {….}
}

class Hero extends ActionCharacter implements CanFight, CanSwim, CanFly {
   public void swim() {…}
   public void fly() {…}
  public void fight() {…}
}
```

Esempio

```
public class Adventure {
  public static void t(CanFight x) { x.fight(); }
  public static void u(CanSwim x) { x.swim(); }
  public static void v(CanFly x) { x.fly(); }
  public static void w(ActionCharacter x) { x. actionfight(); }
  public static void main(String[] args) {
            Hero h = new Hero();
            t(h); // Treat it as a CanFight
            u(h); // Treat it as a CanSwim
            v(h); // Treat it as a CanFly
            w(h); // Treat it as an ActionCharacter
}} ///:~
```

Esempio II - Dov'è il problema ?

```
interface I1 { void f(); }
interface I2 { int f(int i); }
interface I3 { int f(); }
```

```
class C {
   public int f() { return 1; }
}
```

```
class C2 implements I1, I2 {
        public void f() {}
        public int f(int i) { return 1; } // overloaded
}
```

```
class C3 extends C implements I2 { public int f(int i) { return 1; } // overloaded}
class C4 extends C implements I3 {  public int f() { return 1; } //identica}
```

```
class C5 extends C implements I1 {}
class C6 extends C implements I2 {}
class C6 extends C implements I3 {}
```

Errori

InterfaceCollision.java:23: f( ) in C cannot implement f( ) in I1;

attempting to use incompatible return type

found : int required: void

interface I4 extends I1, I3 {}

InterfaceCollision.java:24: interfaces I3 and I1 are incompatible; both define f( ), but with different return type

Considerazioni

Solo un'interfaccia può ereditare interfacce

Solo una classe può implementare interfacce

Ereditare interfacce

```
interface Monster { void menace(); }

interface DangerousMonster extends Monster { void destroy();}

interface Lethal { void kill(); }

interface Vampire extends DangerousMonster, Lethal {
        void drinkBlood();
}

class DragonZilla implements DangerousMonster {
        public void menace() {…}
        public void destroy() {….}
}
```

Attributi di un interfaccia

Un'interfaccia può contenere solo costanti:

```
public interface Months {
        Int JANUARY = 1, FEBRUARY = 2, MARCH = 3,
        APRIL = 4, MAY = 5, JUNE = 6, JULY = 7,
        AUGUST = 8, SEPTEMBER = 9, OCTOBER = 10,
        NOVEMBER = 11, DECEMBER = 12;
} ///:~
```

Tali costanti sono implicitamente final e static

*Inizializzazione delle costanti

In una interfaccia le costanti non possono essere ***blank***, ma l'inizializzazione può avvenire a run-time.

```
public interface RandVals {
   Random rand = new Random();
   int randomInt = rand.nextInt(10);
   long randomLong = rand.nextLong() * 10;
   float randomFloat = rand.nextLong() * 10;
   double randomDouble = rand.nextDouble() * 10;
} ///:~
```

Specifica Interfaccia

- A partire da Java 8, per alleviare il problema delle evoluzioni delle interfacce nelle API, è possibile specificare nelle interfaccia delle implementazioni di default per alcuni dei metodi, così da non forzare le classi implementanti ad offrire un propria implementazione.

```
public interface ResourceLoader {

   Resource load(String resourcePath);

   default Resource load(Path resourcePath) {
      // provide default implementation
   }
}
```

- Una classe può sovrascrivere l'implementazione di default di un metodo semplicemente implementandolo, come avviene normalmente quando si implementa un'interfaccia Java. Qualsiasi implementazione in una classe ha la precedenza sulle implementazioni di default del metodo nell'interfaccia.

Specifica Interfaccia

- Un'interfaccia Java può avere metodi statici con un'implementazione.
- I metodi statici nelle interfacce possono essere utili quando si dispone di alcuni metodi di utilità che si desidera rendere disponibili, che si inseriscono naturalmente in un'interfaccia correlata alla stessa responsabilità.

```
public interface MyInterface {

    public static void print(String text){
        System.out.print(text);
    }
}

…

MyInterface.print("Hello static method!");
```

- Le interfacce Java sono un modo per ottenere il polimorfismo.