Università degli Studi della Campania *Luigi Vanvitelli*

Università di degli Studi della Campania Luigi Vanvitelli Dipartimento di Ingegneria

Programmazione ad Oggetti

a.a. 2020-2021

Interfacce

Docente: Prof. Massimo Ficco

E-mail: *massimo.ficco@unicampania.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*)

Ereditarietà multipla

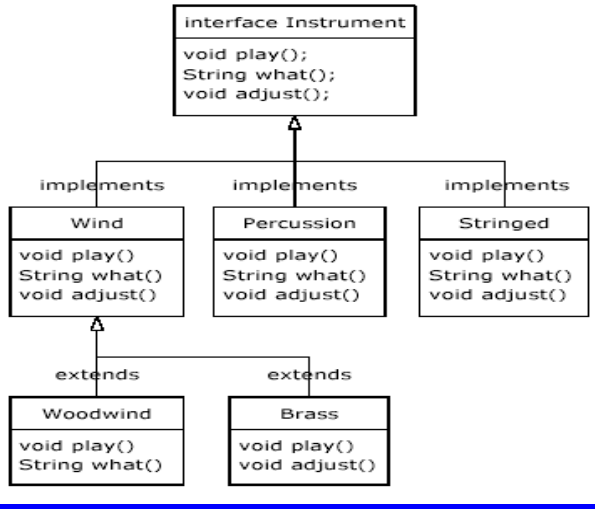
Il c++ supporta l'ereditarietà multipla: ereditare più classi

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***

Esempio



Esempio

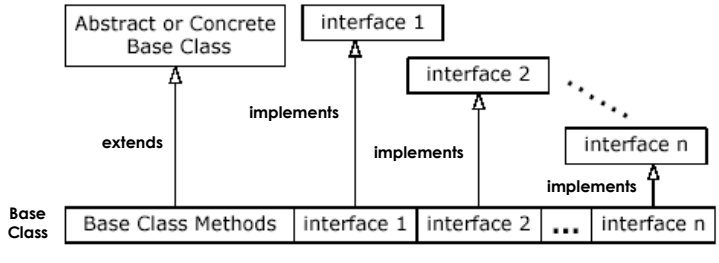
```
interface Instrument {
  // Compile-time constant:
  int i = 5;                    // static & final
  // Cannot have method definitions:
  void play(Note n);            // Automatically public
  String what();
  void adjust();
}

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

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



Esempio V:

```
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 V:

```
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 V:

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 V:

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;
} ///:~
```

V:

A cura del

Prof. Massimo Ficco

e del

Prof. Salvatore Venticinque