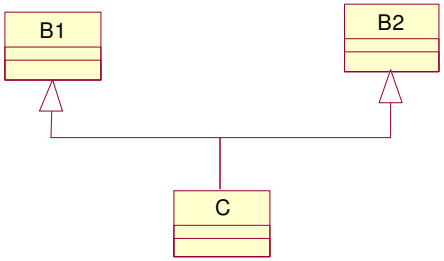
1

Multiple inheritance



2

Multiple Inheritance

```
#include <iostream.h>
class B1 {
  public:
    B1() {cout << "calling constructor of B1" << endl;}
    void f1() { cout << "calling f1 in B1" << endl; }
    };
class B2{
  public:
    B2() {cout << "calling constructor of B2" << endl;}
    void f2() { cout << "calling f2 in B2" << endl; }
};
class C:  public B1,  public B2 {
  public:
    C() {cout << "calling constructor of C" << endl;}
};
```

3

chiamata dei metodi

```
int main()
{
        B1 * b1=new B1();
        b1->f1();

        B2 * b2=new B2();
        b2->f2();

        C * c=new C();
        c->f1();
        c->f2();
}
```

```
calling constructor of B1
calling f1 in B1

calling constructor of B2
calling f2 in B2

calling constructor of B1
calling constructor of B2
calling constructor of C

calling f1 in B1
calling f2 in B2
```

4

Pro e contro

Comoda, perchè in C++
le gerarchie di classi sono
DISGIUNTE

Complessa...

5

In Java...

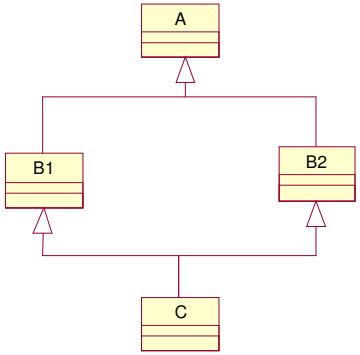
Come molte altre cose che complicano la vita,
in Java NON C'E'! ma...

...ci sono le INTERFACCE
(classi COMPLETAMENTE astratte, senza variabili di istanza)

```
interface Storable {
   setValue(int k);
   int getValue();
}
interface Counter {
   setCount(int k);
   int getCount();
}
class A extends B implements Storable, Counter {
   ...
}
... Storable x=new A(); ...
```

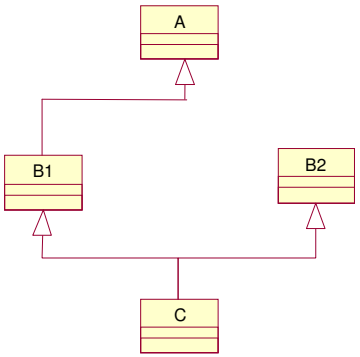
6

"Diamond" Multiple Inheritance



7

Cominciamo con le cose semplici...



8

Quali metodi ha C?

```
class A {
  public:
    A() {cout << "calling constructor of A" << endl;}
     void f() { cout << "calling f in A" << endl; }
};
class B1 : public A {
  public:
    B1() {cout << "calling constructor of B1" << endl;}
    void f1() { cout << "calling f1 in B1" << endl; }
    };
class B2{
  public:
    B2() {cout << "calling constructor of B2" << endl;}
    void f2() { cout << "calling f2 in B2" << endl; }
};
class C:  public B2,  public B1 {
  public:
    C() {cout << "calling constructor of C" << endl;}
};
```

9

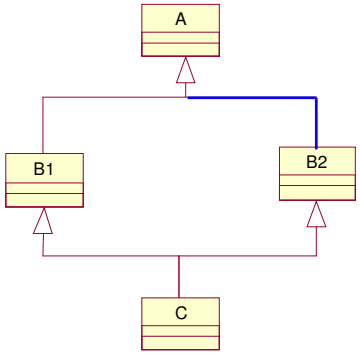
Esempio

```
int main()
{
      B1 * a1=new B1();
      b1->f1();
      b1->f();
      B2 * b2=new B2();
      b2->f2();
      C * c=new C();
      c->f();
      c->f1();
      c->f2();
}
```

```
calling constructor of A
calling constructor of B1
calling f1 in B1
calling f in A

calling constructor of B2
calling f2 in B2

calling constructor of B2
calling constructor of A
calling constructor of B1
calling constructor of C

calling f in A
calling f1 in B1
calling f2 in B2
```

10

...e rendiamole complesse



Che succede quando accedo da C a A::f() ?

11

!Error!

```
class B2: public A{
  public:
    B2() {cout << "calling constructor of B2" << endl;}
    void f2() { cout << "calling f2 in B2" << endl; }
};
```

Compiler error

[C++ Error] MultipleInheritance.cpp(31): E2014
Member is ambiguous: 'A::f' and 'A::f'.

12

Fix

```
class B1 : virtual public A {
  public:
    B1() {cout << "calling constructor of B1" << endl;}
    void f1() { cout << "calling f1 in B1" << endl; }
    };
class B2: virtual public A{
  public:
    B2() {cout << "calling constructor of B2" << endl;}
    void f2() { cout << "calling f2 in B2" << endl; }
};
```

13

Funziona. Ma che vuol dire?

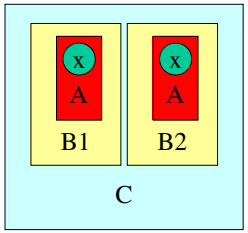
```
int main()
{
      B1 * b1=new B1();
      b1->f1();
      b1->f();

      B2 * b2=new B2();
      b2->f2();
      b2->f();

      C * c=new C();

      c->f();
      c->f1();
      c->f2();
}
```

```
calling constructor of A
calling constructor of B1
calling f1 in B1
calling f in A

calling constructor of A
calling constructor of B2
calling f2 in B2
calling f in A

calling constructor of A
calling constructor of B1
calling constructor of B2
calling constructor of C

calling f in A
calling f1 in B1
calling f2 in B2
```

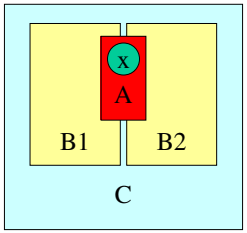
14

Funziona. Ma che vuol dire?

```
class A{public: int x;}
class B1:public A{…};
class B2:public A{…};
class C:public B1,B2{…};
```




```
class A{public: int x;}
class B1:virtual public A{…};
class B2:virtual public A{…};
class C:public B1,B2{…};
```



15

operatore di scope (1)

```
#include <iostream.h>
class A { public: int x; };
class A1 :  public A {
  public:
    void f1(int k) {x=k; }
    };
class A2:  public A{
  public:
    void f2(int k) {x=k;}
};
class A3:  public A1,  public A2 {};
int main()
{
       A3 * a3=new A3();
       a3->f1(3);
       a3->f2(6);
       cout<<a3->A2::x<<"==="<<a3->A1::x<<endl;
      return 0;
}
```

6===3

Operatore di scope

16

operatore di scope (2)

```
#include <iostream.h>
class A { public: int x; };
class A1 :  virtual public A {
  public:
    void f1(int k) {x=k; }
    };
class A2:  virtual public A{
  public:
    void f2(int k) {x=k;}
};
class A3:  public A1,  public A2 {};
int main()
{
       A3 * a3=new A3();
       a3->f1(3);
       a3->f2(6);
       cout<<a3->A2::x<<"==="<<a3->A1::x<<endl;
      return 0;
}
```

6===6

17

operatore di scope (3)

```
#include <iostream.h>
class A { public: int x; };
class A1 :  virtual public A {
  public:
    void f1(int k) {x=k; }
    };
class A2:  public A{
  public:
    void f2(int k) {x=k;}
};
class A3:  public A1,  public A2 {};
int main()
{
       A3 * a3=new A3();
       a3->f1(3);
       a3->f2(6);
       cout<<a3->A2::x<<"==="<<a3->A1::x<<endl;
      return 0;
}
```

6===3

18

Inheritance vs. delegation

19

Inheritance...

```
#include <iostream.h>
class A {
    int x;
    public:
      void setX(int k){x=k;}
      int getX(){return x;}
};
class B:public A {
    int y;
    public:
      void setY(int k){y=k;}
      int getY(){return y;}
};
```

```
int main()
{
     B b;
     b.setX(3);   b.setY(4);
     cout<<b.getY()<<" "<<b.getX();
}
```

20

```
#include <iostream.h>
class A {
    int x;
    public:
      void setX(int k){x=k;}
      int getX(){return x;}
};

class B {
    A delegate;
    int y;
    public:
      void setY(int k){
        y=k;}
      int getY(){
        return y;}
      void setX(int k){
        delegate.setX(k);}
      int getX(){
        return delegate.getX(k);}

};
```

...vs. Delegation

```
int main()
{
  B b;
  b.setX(3);   b.setY(4);
  cout<<b.getY()<<" "<<b.getX();
}
```

21

Inheritance...

```
class A {
   int x;
   public void setX(int k){x=k;}
   public int getX(){return x;}
};
class B extends A {
   int y;
   public void setY(int k){y=k;}
   public int getY(){return y;}
};
```

```
public static void main(String[] args) {
   B b=new B();
   b.setX(3);
   b.setY(4);
   System.out.println(b.getY()+" "+b.getX());
 }
```

22

...vs. Delegation

```
class A {
   int x;
   public void setX(int k){x=k;}
   public int getX(){return x;}
};
class B {
   private A a=new A();
   int y;
   public void setY(int k){y=k;}
   public int getY(){return y;}
   public void setX(int k){a.setX(k);}
   public int getX(){return a.getX();}
};
```

```
public static void main(String[] args) {
   B b=new B();
   b.setX(3);
   b.setY(4);
   System.out.println(b.getY()+" "+b.getX());
 }
```