Inheritance

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Miri Ben-Nissan
- Version 3: Dr. Erel Segal-Halevi

	C++	Java
Keyword	:	“extends”
Access to base	public, protected or private	public
Poly-morphic	Only if requested	Always
Multiple parents	Yes (useful mainly for interfaces)	No (except interfaces)
Interface keyword	"class"	“interface”
Universal base	None	Object

Person

```
class Person
{
private:
   std::string _name;
   int _id;
   static const int NO_ID_VAL= -1;
public:
   Person (const std::string& name, int id);
   void changeName(const string& name);
   void changeId(int id);
   std::string getName() const;
   int getId() const;
} ;
```

Programmer class

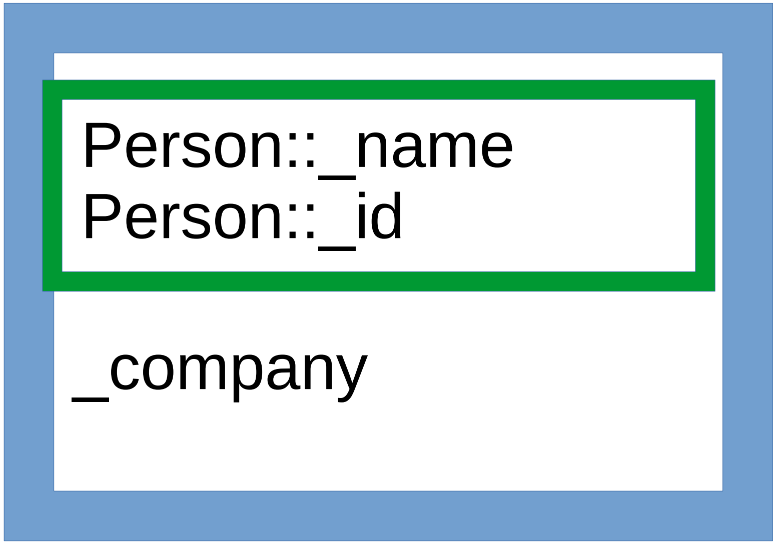
```
#include "Person.hpp"
class Programmer : public Person
{
    string _company;
public:
    string getCompany() const {...}
    void setCompany(string c) {...}
    …
}
```

Base class

Derived class

Inheritance – under the hood

Every object of class Programmer contains a hidden field of class Person.



Inheritance – under the hood

```
#include "Person.hpp"
class Programmer
{
public:
   Person person;
private:
   std::string _company;
public:
   string getCompany() const {...}
   void setCompany(string c) {...}
   …
;
```

Objects of Programmer can use Person’s methods

```
int main()
{
  Programmer yoram("Yoram",1226611,"N.G.C ltd.");

  cout << yoram.getCompany() << endl;
  yoram.changeCompany("Microsoft");

  cout << yoram.getName()   << " " <<
         yoram.getId()      << endl;
  yoram.changeName("Yori");
  yoram.changeId(2266110);  // equivalent to:
  yoram.Person::changeId(2266110);
}
```

Objects of Programmer can use Person’s methods

```
int main()
{
  Programmer yoram("Yoram",1226611,"N.G.C
ltd.");
  Person yoram2 = yoram; // slicing
  // equivalent to:
  Person yoram2 = (Person)yoram;

  cout << yoram2.getName()  << " " <<
         yoram2.getId()     << endl;
}
```

Programmer class - constructor

```
#include "Programmer.hpp"

Programmer::Programmer
         (const std::string& name,
          int id,
          const std::string& company) :
  Person(name, id), _company(company)
{
   // EMPTY  从  Considered elegant
}
```

protected

- Class members that should be accessible by subclasses only are declared as protected.
- To allow class Programmer to access the members of class Person, define:

```
class Person
{
protected:
    std::string _name;
    int _id;
    static const int NO_ID_VAL= -1;
public:
    ...
```

public, protected and private inheritance

A base class also has an access modifier:

```
class Programmer : public Person
```

Default for structs

or

```
class Programmer : protected Person
```

or

```
class Programmer : private Person
```

Default for classes

- This modifier relates to the **hidden object** of type Person that is contained in Programmer.
- Private inheritance is barely used in practice, *but* you might get it by mistake if you forget to write "public" (since it is the default).

Objects of Programmer can use Person’s methods

```
int main()
{
    Programmer yoram("Yoram",1226611,"N.G.C ltd.");

    cout << yoram.getCompany() << endl;
    yoram.changeCompany("Microsoft");

// This doesn't compile with private inheritance:
    // cout << yoram.getName()  << " " <<
    //          yoram.getId()     << endl;
    // yoram.changeName("Yori");
    // yoram.changeId(2266110);
    // yoram.Person::changeId(2266110);
}
```

C-tor & D-tor order of execution

C-tor & D-tor order of execution

```
int main() {B b(1,2);}
```

What will be the output?

```
class A {
  int _a;
 public:
  A(int a) : _a(a) { cout << "A ctor\n"; }
  ~A()             { cout << "A dtor\n"; }
};

class B : public A {
  int _b;
 public:
  B(int a, int b) : A(a), _b(b) { cout << “B ctor\n"; }
  ~B()                          { cout << “B dtor\n“; }
};
```

C-tor & D-tor order of execution

1. Constructor of the base class is executed
2. Constructor of the class itself is executed

C-tor & D-tor order of execution

1. Constructor of the base class is executed
    1. First members in initialization list
    2. Then body
2. Constructor of the class itself is executed
    1. First members in initialization list
    2. Then body

C-tor & D-tor order of execution

```
int main()
{
    B b(1,2);
}
```

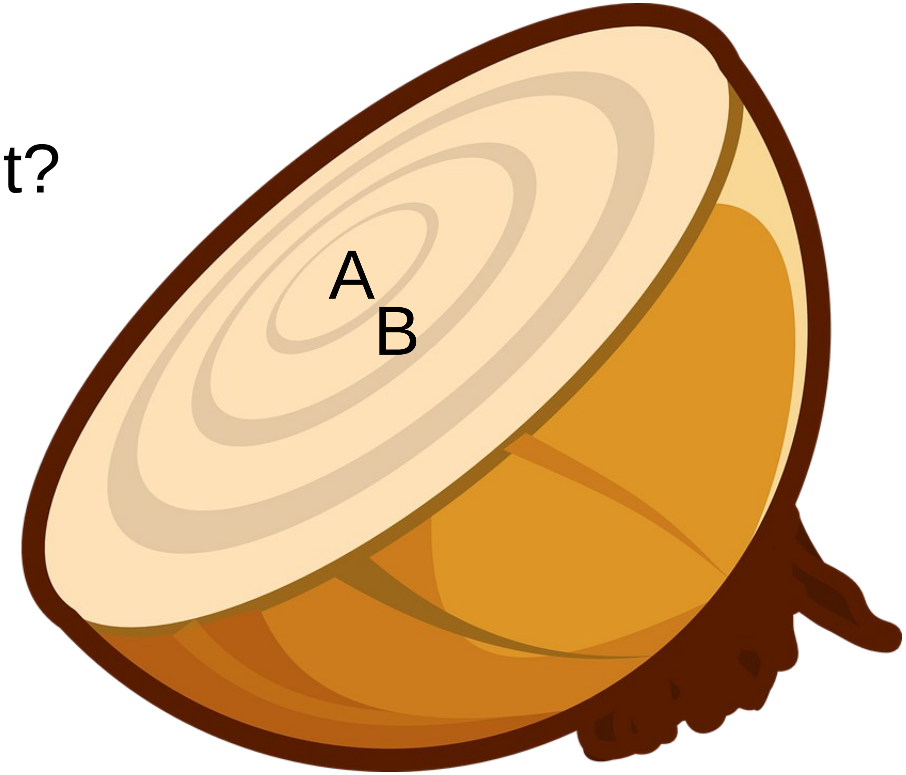
What will be the output?

A ctor

B ctor

B dtor

A dtor



C-tor & D-tor order of execution - demo

Either view **folder 1**

Or put the following code in https://godbolt.org/

```
struct A {
    int i;
    A() { i = 555; }
    ~A() { i = 666; }
};

struct B: public A {
    int j;
    B() { j = 777; }
    ~B() { j = 888; }
};

int main() {
    B b;
}
```

Overriding

Person

```
class Person
{
    ...
   void output(std::ostream& os) const;
    ...
} ;
```

Programmer class – Override

```
#include "Person.hpp"
class Programmer : public Person
{
    ...
   void output(std::ostream& os) const;
    ...
};
```

Overridden member functions (folder 2)

```
void Person::output(std::ostream& os) const {
   os << "{";
   if(_name != "") os << " name: " << _name;
   if(_id != NO_ID_VAL) os << " ID: " << _id;
   os << '}';
}
```

```
void Programmer::output(std::ostream& os) const {
    Person::output(os);
    os << '-' << _company;
}
```

Explicit Operator=

```
Person& Person::operator=(const Person& other)
{
      ...
   return *this;
}
Programmer& Programmer::operator=(const
Programmer& other)
{
    Person::operator=(other);
    company = other.company;
….
    return *this;
}
```