Low Level Design (LLD)

Part 2

Constructors

Unlike normal member functions, constructors have specific rules for how they must be named:

- Constructors must have the same name as the class (with the same capitalization).
- Constructors have no return type **(not even void)**.

Destructors

Classes have another type of special member function that is called automatically when an object is destroyed. This function is called a destructor.

Destructors are designed to allow a class to do any necessary clean up before an object of the class is destroyed.

Like constructors, destructors have specific naming rules:

- The destructor must have the same name as the class, preceded by a tilde (~).
- The destructor can not take arguments.
- The destructor has no return type.

Example

```
class A {
public:
    A(int a) : a_(a) {}
    ~A() {}

private:
    int a_:
};
```

Constructor

Destructor

Constructor member initializer lists

Let’s go to VS Code!

Constructor member initializer lists

- To have a constructor initialize members, we do so using a member initializer list (often called a “member initialization list”).

- The members in a member initializer list are always initialized in the order in which they are defined inside the class (not in the order they are defined in the member initializer list).

- The bodies of constructors functions are most often left empty. However, because the statements in the body of the constructor execute after the member initializer list has executed, we can add statements to do any other setup tasks required.

Let’s come back to Inheritance!

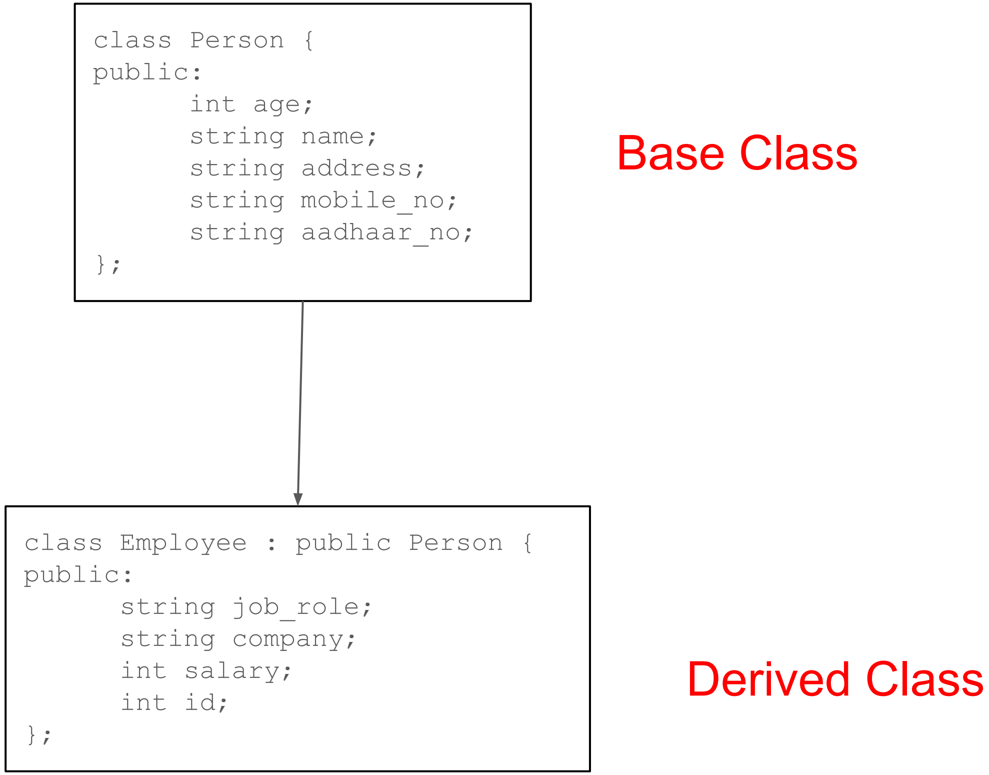
Inheritance

```
class Person {
public:
     int age;
     string name;
     string address;
     string mobile_no;
     string aadhaar_no;
};
```

```
class Employee : public Person {
public:
     string job_role;
     string company;
     int salary;
     int id;
};
```

```
class SoftwareEngineer : public Employee {
public:
     string team;
     string tech_stack;
     string language_expertise;
     int level;
};
```

Inheritance Terminology



```
class Person {
public:
      int age;
      string name;
      string address;
      string mobile_no;
      string aadhaar_no;
};
```

```
class Employee : public Person {
public:
      string job_role;
      string company;
      int salary;
      int id;
};
```

Construct Inherited Class

Let’s go to VS Code!

Construct / Destruct Inherited Class

- Most base class is initialized first.
- Most derived class is destructed first.

Inheritance and access specifiers

- Public: can be accessed by anybody
- Private: can only be accessed by class members
- **Protected: can be accessed by class members, and derived classes.**

Different kinds of Inheritance

```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

```
class B : public A
{
    a_public_;

    ~~a_private_~~

    a_protected_
};
```

```
int main() {
    B b;

    b.a_public_;

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

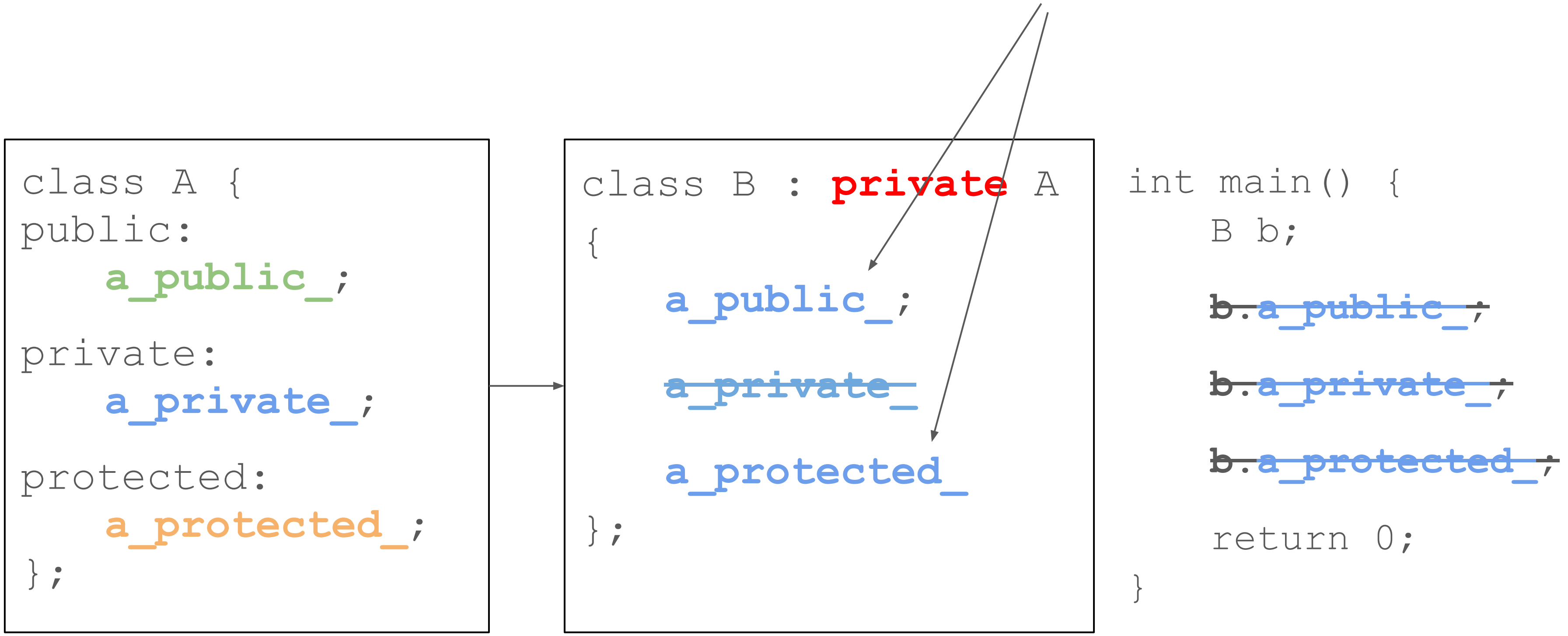
Different kinds of Inheritance

```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

```
class B : private A
{
    a_public_;

    ~~a_private_~~

    a_protected_
};
```

```
int main() {
    B b;

    ~~b.a_public_;~~

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

Different kinds of Inheritance



```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

```
class B : private A
{
    a_public_;

    ~~a_private_~~

    a_protected_
};
```

```
int main() {
    B b;

    ~~b.a_public_;~~

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

Different kinds of Inheritance

```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

$\rightarrow$

```
class B : protected A
{

    a_public_;

    ~~a_private_~~

    a_protected_
};
```

```
int main() {
    B b;

    ~~b.a_public_;~~

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

Different kinds of Inheritance

Accessible, but behave like protected members of class B

```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

```
class B : protected A
{
    a_public_;

    ~~a_private_~~

    a_protected_
};
```

```
int main() {
    B b;

    ~~b.a_public_;~~

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

Most Practical Inheritance

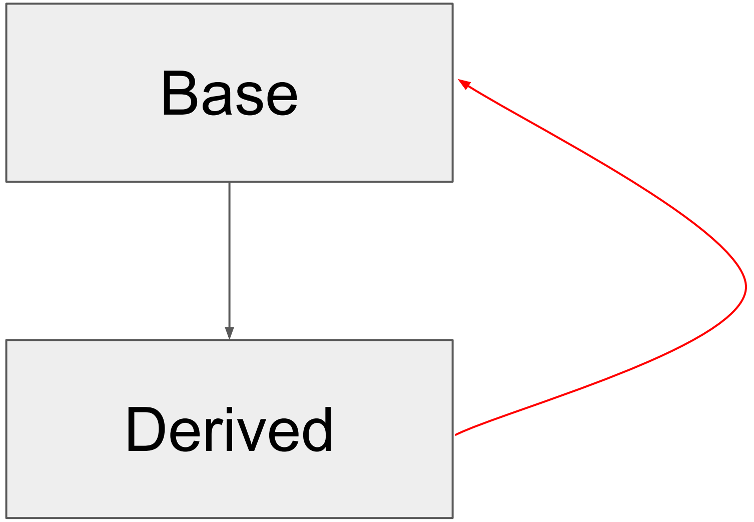
Remember others only for theory purposes :P

```
class A {
public:
    a_public_;

private:
    a_private_;

protected:
    a_protected_;
};
```

```
class B : public A
{
    a_public_;

    ~~a_private_~~

    a_protected_
};
```

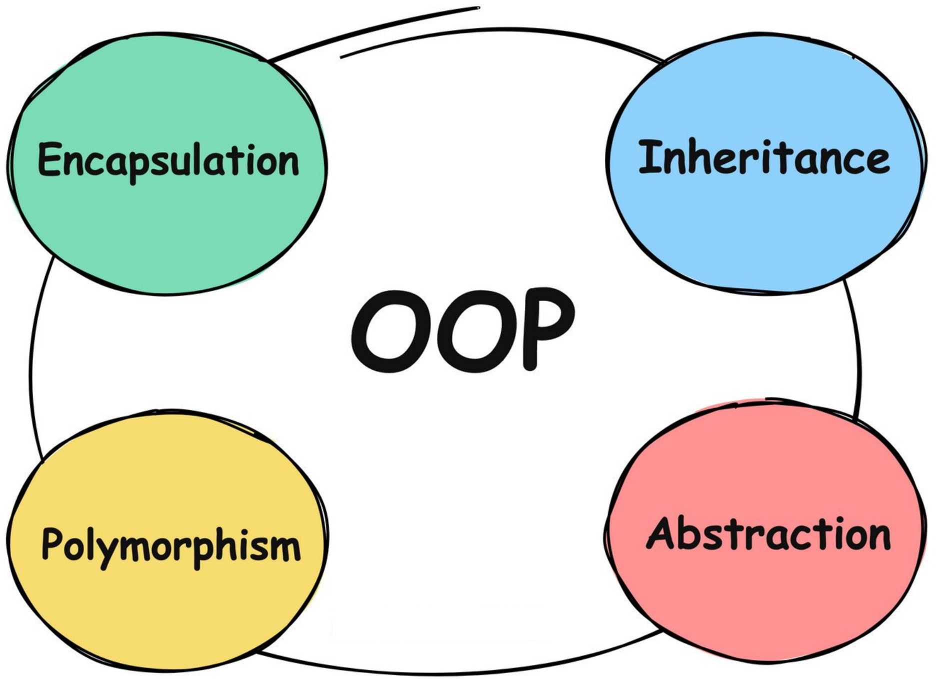
```
int main() {
    B b;

    b.a_public_;

    ~~b.a_private_;~~

    ~~b.a_protected_;~~

    return 0;
}
```

Best practice to construct inherited classes



From the constructor of Derived class, call the constructor of Base class.

The corresponding constructors will take care of initializing their class members.


Encapsulation
Inheritance
OOP
Polymorphism
Abstraction

Polymorphism

The word “polymorphism” means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Polymorphism

Let’s go to VS Code!

What’s next?

- Advance OOPs concepts
  - Polymorphism
  - Virtual Functions
  - Abstractions

Thank You!