

# Low Level Design (LLD)

Part 3

# Inheritance

```
int main() {  
    Derived D;  
  
    return 0;  
}
```

# Inheritance

```
int main() {  
    Derived D;  
  
    return 0;  
}
```

public:

A();

B();

C();

public

public:

A();

E();

F();

Base

Type of  
Inheritance

Derived

# Inheritance

```
int main() {  
    Derived D;  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

```
public:  
    A();  
    D();  
    E();
```

# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

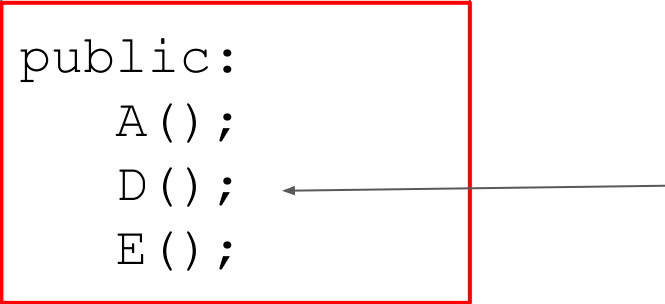
```
public:  
    A();  
    D();  
    E();
```

# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

```
public:  
    A();  
    D();  
    E();
```

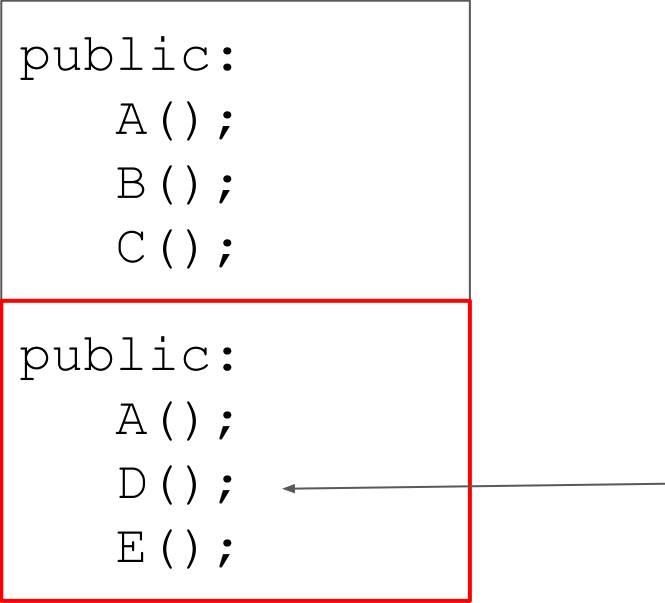


# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    D.B();  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

```
public:  
    A();  
    D();  
    E();
```



# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    D.B();  
  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

D();

E();



# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    D.B();  
  
    D.A();  
  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

D();

E();

# Inheritance

```
int main() {  
    Derived D;  
  
    D.D();  
  
    D.B();  
  
    D.A();  
  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

D();

E();

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    return 0;  
}
```

```
public:
```

```
    A();
```

```
    B();
```

```
    C();
```

```
public:
```

```
    A();
```

```
    E();
```

```
    F();
```

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
  
    return 0;  
}
```

```
public:  
    A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

E();

F();

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
  
    B.A();  
  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

E();

F();

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
    B.A();  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

E();

F();



# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
    B.A();  
    B.E();  
    return 0;  
}
```

public:

A();

B();

C();

public:

A();

E();

F();

# Inheritance

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.B();  
    B.A();  
    B.E();  
    return 0;  
}
```

Compile Error

public:

A();

B();

C();

public:

A();

E();

F();

# Polymorphism

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    return 0;  
}
```

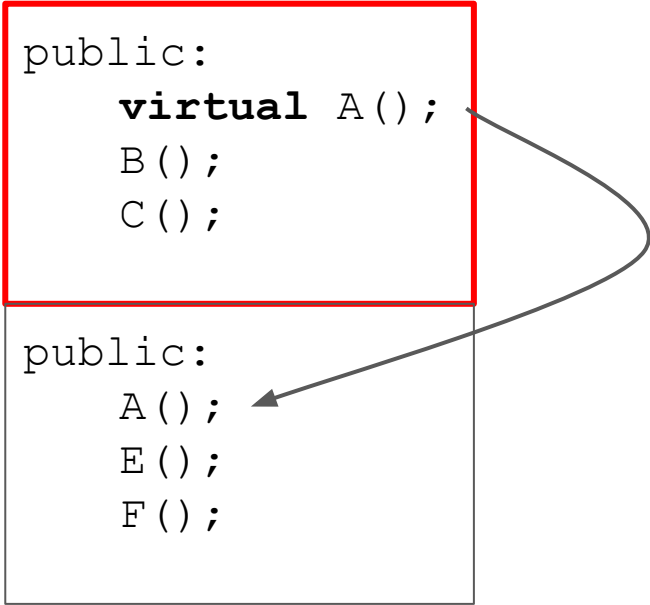
```
public:  
    virtual A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```

# Polymorphism

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    return 0;  
}
```

```
public:  
    virtual A();  
    B();  
    C();
```



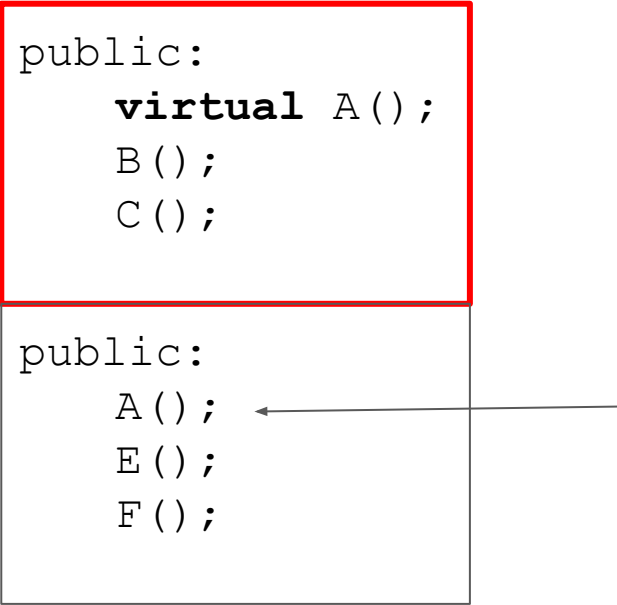
```
public:  
    A();  
    E();  
    F();
```

# Polymorphism

```
int main() {  
    Derived D;  
  
    Base &B = D;  
  
    B.A();  
  
    return 0;  
}
```

```
public:  
    virtual A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```



# Override and Final

In a member function declaration or definition, **override** specifier ensures that the function is virtual and is overriding a virtual function from a base class. The program is ill-formed (a compile-time error is generated) if this is not true.

When used in a virtual function declaration or definition, **final** specifier ensures that the function is virtual and specifies that it may not be overridden by derived classes. The program is ill-formed (a compile-time error is generated) otherwise.

# Virtual Destructors

*Let's go to VS Code!*

# Best Practices

- Use the virtual keyword on virtual functions in a base class.
- Use the override specifier (but not the virtual keyword) on override functions in derived classes. This includes virtual destructors.
- Whenever you are dealing with inheritance, you should make any explicit destructors virtual.



```
int main() {  
    Derived D;  
  
    Base B = D;  
  
    B.A();  
  
    return 0;  
}
```

```
public:  
    virtual A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```

# Object Slicing

```
int main() {  
    Derived D;  
  
    Base B = D;  
  
    B.A();  
  
    return 0;  
}
```

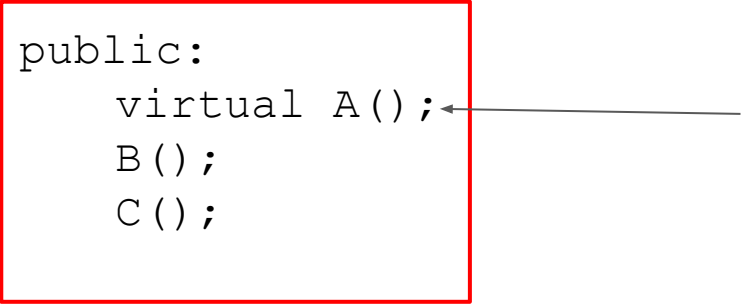
```
public:  
    virtual A();  
    B();  
    C();
```

```
public:  
    A();  
    E();  
    F();
```

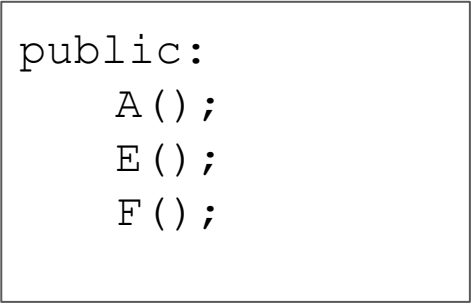
# Object Slicing

```
int main() {  
    Derived D;  
  
    Base B = D;  
  
    B.A();  
  
    return 0;  
}
```

```
public:  
    virtual A();  
    B();  
    C();
```



```
public:  
    A();  
    E();  
    F();
```



# Pure virtual function

A special kind of virtual function that has no body at all!

A pure virtual function simply acts as a placeholder that is meant to be redefined by derived classes.

# Abstract Base Class

Any class with one or more pure virtual functions becomes an abstract base class, which means that **it can not be instantiated!**

# Interface Class

An **interface class** is a class that has no member variables, and where all of the functions are pure virtual!

Interfaces are useful when you want to define the functionality that derived classes must implement, but leave the details of how the derived class implements that functionality entirely up to the derived class.

Thank You!