1. Each **class** controls how its members are initialized
2. The **base class** is initialized first, and then the **members** of the **derived class** are initialized in the order in which they are **declared** in the class.
3. If a base class defines a static member, there is only one such member defined for the entire hierarchy.
4. we can prevent a class from being used as a base by following the class name with **final**.
5. We must define every virtual function, regardless of whether it is used, because the compiler has no way to determine whether a virtual function is used
6. A **derived-class** function that **overrides** an inherited **virtual function** must have exactly the same parameter type(s) as the **base-class** function that it overrides.
7. With one exception, the return type of a virtual in the derived class also must match the return type of the function from the base class. The exception applies to **virtuals** that return a reference (or pointer) to types that are themselves related by inheritance. Example
8. A function that is virtual in a base class is implicitly **virtual** in its derived classes. When a derived class **overrides** a **virtual**, the **parameters** in the base and derived classes **must match exactly**
9. We can designate a function as **final**. Any attempt to override a function that has been defined as **final** will be flagged as an error.
10. **final** and **override** specifiers appear after the parameter list (including any const or reference qualifiers) and after a trailing return.
11. If a class uses a default argument, the value that is used is the one defined by the **static type** through which the function is called.
12. **Virtual functions** that have default arguments should use the same argument values in the base and derived classes.
13. In some cases, we want to prevent dynamic binding of a call to a virtual function; We can use the scope operator to force that call to use a particular version of that virtual.
14. The **=0** may appear only on the declaration of a virtual function in the class body.
15. We can provide a definition for a **purevirtual**. However, the function body must be defined outside the class.
16. A derived class member or friend may access the **protected** members of the base class **only** through a derived object. The derived class has no special access to the **protected** members of **base-class** objects. [Example](Object-Oriented%20Programming.cpp)
17. Accessibility of Derived-to-Base Conversion
    1. **User code** may use the **derived-to-base** conversion **only if** D inherits **publicly** from B. User code may not use the conversion if D inherits from B using either protected or private.
    2. **Member functions** and **friends of D** can use the conversion to B regardless of how D inherits from B. The **derived-to-base conversion** to a direct base class is **always accessible** to members and friends of a derived class.
    3. **Member functions** and **friends of classes derived from D** may use the **derived-to-base conversion** if D inherits from B either **public** or **protected**. Such code may not use the conversion if B inherits privately from B.
18. Each class controls access to its own members. The access includes access to Base objects that are embedded in an object of type derived from Base. [Example](file:///G:\Code\C++\C++primer\Note\Object-Oriented%20Programming.cpp)
19. We can change the **access level** of a **name** that derived class inherits by providing a **using** declaration. Access to a name specified in a **using** declaration depends on the access specifier preceding the **using** declaration. [Example](Object-Oriented%20Programming.cpp)
20. A derived class may provide a using declaration only for names it is permitted to access.
21. By default, a derived class defined with the **class** keyword has private inheritance; a derived class defined with **struct** has public inheritance;
22. The only differences of classes defined using the struct keyword and those defined using class are the default access specifier for members and the default derivation access specifier. There are no other distinctions.
23. A derived-class member with the same name as a member of the base class hides direct use of the base-class member.
24. Aside from overriding inherited **virtual** functions, a derived class usually **should not** reuse names defined in its base class.
25. If a ctor or dtor calls a **virtual**, the version that is run is the one **corresponding to** the type of the **ctor** or **dtor** itself.
26. Unlike **using** declarations for ordinary members, a ctor **using** declaration does not change the access level of the inherited ctor(s).
27. Moreover, a **using** declaration can’t specify explicit or constexpr.
28. If a base-class ctor has default arguments, those arguments are not inherited. Instead, the derived class gets multiple inherited ctors in which each parameter with a default argument is successively omitted. For example, if the base has a ctor with two parameters, the second of which has a default, the derived class will obtain two ctors: one with both parameters (and on default argument) and a second ctor with a single parameter corresponding to the left-most, non-defaulted parameter in the base class. [Example](Object-Oriented%20Programming.cpp)
29. If a base class has several ctors, then with two exceptions, the derived class inherits each of the ctors from its base class.
    1. The first exception is that a derived class can inherit some ctors and define its own versions of the other ctors. If the derived class defines a ctor with the **same parameters** as a ctor in the base, then that ctor is **not** inherited. The one defined in the derived class is used in place of the inherited ctor.
    2. The second exception is that the **default**, **copy**, and **move** ctor are not inherited. These ctors are synthesized using the normal roles. An **inherited ctor** is **not** treated as a user-defined ctor. Therefor, a class that contains only inherited ctors will have a synthesized default ctor.