



Recitation Class 11 for VG101

Date: 2012 / 12 / 10

Wang Qian

Inheritance

- When should we use inheritance?
 - To show the “*is-a*” relation instead of “*has-a*” relation.
 - For the “*has-a*” relation, we can use implement of composition.
- Principles:
 - There are five important SOLID principles in OOP.
 - Today, I want you to understand two of them.
 - **Liskov Substitution Principle (LSP)**
 - **Dependency Inversion Principle (DIP)**

Liskov Substitution Principle

- Liskov Substitution Principle:
 - Let B be the base type and D be the derived type of B.
 - $\forall x \in B: f(x) \Rightarrow \forall y \in D: f(y)$
- Classical Problem
 - Is it a good idea to derive Circle from Ellipse?
 - Is it a good idea to derive Ostrich from Bird?
 - Use LSP to explain the reason.

Derive a Class

- How to implement inheritance in C++?
 - `class DerivedClass : mode BaseClass`
- Note:
 - Derived class stored the members and the member functions of the base class, though the accessibility depends on the mode.
 - Derived class should have its own constructor.
 - You can add more members or member functions to the derived class if needed.

Inheritance Mode

inheritance mode	base public member	base protected member	Base private member
public	derived public member	derived protected member	not accessible
protected	derived protected member	derived protected member	not accessible
private	derived private member	derived private member	not accessible

- What kind of member is accessible outside both the base class and the derived class?
 - public member only

Initializer List

- Initializer list can only be used in the constructor.
 - `BaseClass::BaseClass(type x)`

```
{  
    mem = x  
}
```
 - `DerivedClass::DerivedClass (type x,type y):BaseClass(y),mem(x)`

```
{  
}
```
 - `DerivedClass::DerivedClass (type x,type y):BaseClass(y)`

```
{  
    mem = x;  
}
```

Reference and Pointer

- Remember the following relations:
 - A pointer of the base class can point to its derived object.
 - A reference of the base class can refer its derived object.
 - A pointer of the derived class cannot point to its base object.
 - A reference of the derived class cannot refer its base object.
- This can be used to explained a function in the lecture:

```
void tune(Instrument &i) {  
    // ...  
    i.play();  
}  
int main() {  
    Wind flute;
```

- Pointer or reference of the base class cannot use the member function added in its derived object.

Polymorphism

- Two tasks:
 - Redefine the member function of the base class in the derived class.
 - Use the virtual method.
- Implement of the virtual method:
 - Add the key word “virtual” just at the beginning of the member function prototype in the base class.
 - The characteristic of virtual method will appear for the pointer or the reference of an object.

Virtual Method

- Characteristic:
 - Without the key word “virtual”, program will choose the running member function according to the class of the reference or the pointer itself.
 - With the key word “virtual”, program will choose the running member function according to the class of the object associating with the reference or the pointer.
 - Use this characteristic to explain why virtual destructor is needed?
 - Consider, in the example code, what will happen when such a statement as “delete member[i];” is implemented.

Virtual Method

- This will be a clearer example.
 - BaseClass object1;
DerivedClass object2;
BaseClass &ref1 = object1;
BaseClass &ref2 = object2;
ref1.memFunc(); // Always use BaseClass::memFunc
ref2.memFunc(); // Use DerivedClass::memFunc if virtual

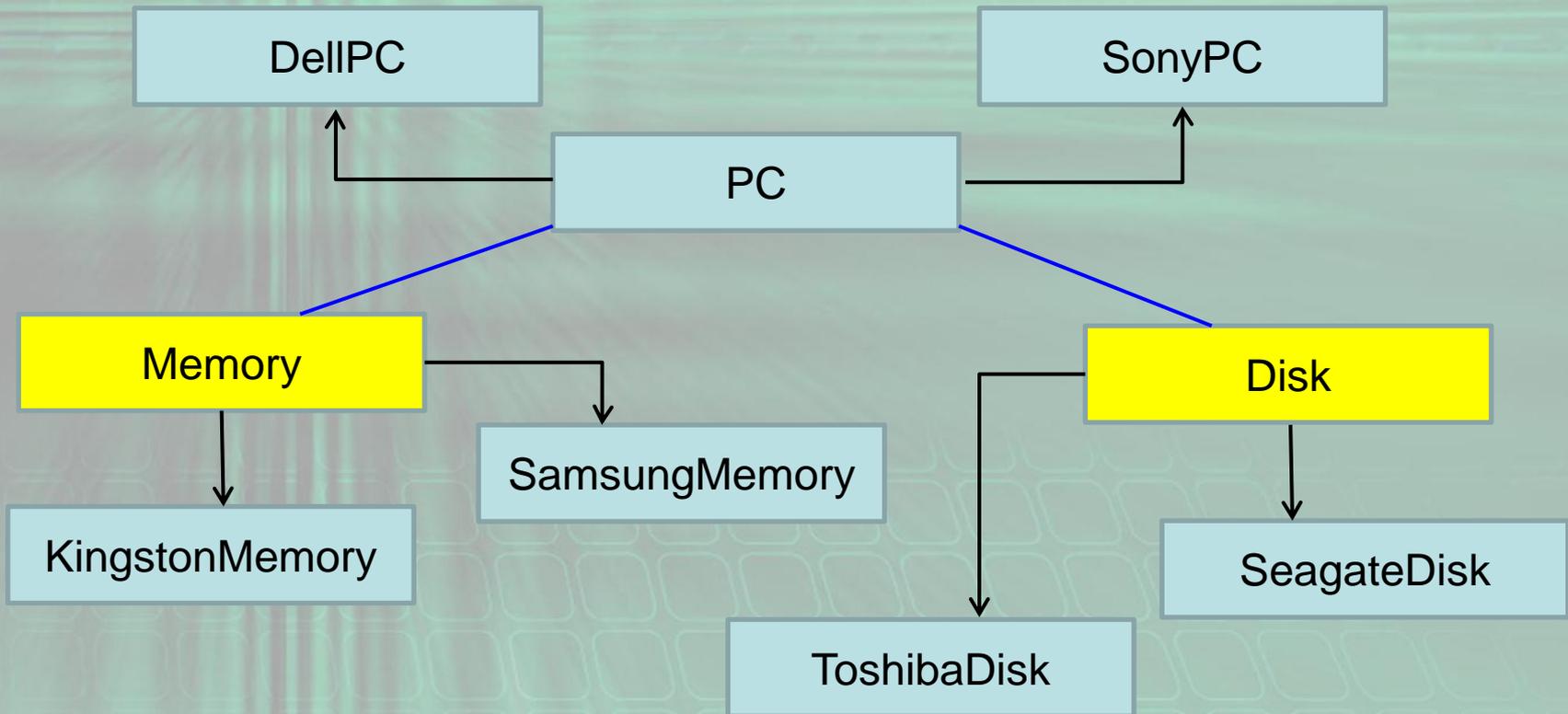
Array of Base Class Pointer

- Impossible to store data with different types in one array.
- But, base pointer can point to different derived class.
 - E.g.
 - `BaseClass *obj [MAX];`
 - `obj[i] = new DerivedClass;`
 - `(*obj[i]).memFunc();`
 - `obj[i]->memFunc();`

Dependency Inversion Principle

- Dependency Inversion Principle:
 - Detailed type should depend on abstract type, but abstract type should not depend on detailed type.
 - Either high-level and low-level type should not depend on the other, but should depend on abstract type instead.
- For example:
 - Show the relationship of the following classes: DellPC, SonyPC, PC, KingstonMemory, SamsungMemory, Memory, SeagateDisk, ToshibaDisk, Disk

Dependency Inversion Principle



- Black arrows mean inheritance.
- Blue lines mean implement of composition.
- Yellow blocks mean the abstract classes.

Abstract Base Class

- It is a kind of base class containing pure virtual function.
 - The virtual function ended by “=0”.
 - The pure virtual functions can only be used in base class.
- An object of abstract base class can never be built up.
 - For example, in the code I provided, Employee is an abstract base class. Therefore, you cannot use declare an object with the type of Employee.