Multiple Inheritance

Chapter 13

Objectives

- Examine variations on inheritance
- Learn how to disambiguate
- The importance of virtual base classes
- Model design considerations
  - Sample code is in chapter folder.

Types of inheritance

- Single Inheritance
  - In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.
- Multiple Inheritance
  - Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one subclass is inherited from more than one base class.
- Multipath inheritance
  - A derived class with two base classes and these two base classes have one common base class is called multipath inheritance.
  - Ambiguity can arise in this type of inheritance.

Public, Protected, and Private inheritance

- Public Inheritance
  - Makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.
- Protected Inheritance
  - Makes the public and protected members of the base class protected in the derived class.
- Private Inheritance
  - Makes the public and protected members of the base class private in the derived class.
- Public inheritance models ‘IS-A’ relationship, whereas both protected and private inheritance represent an ‘IMPLEMENTED-IN-TERMS-OF’ relationship.

Model Design Considerations

- Consider using at most one base class that represents a true type
  - class name is a noun
  - Consider using templates for generality and reuseability
- Use a class that represents an interface type for all others
  - class name describes a “can do” contract
  - All methods are pure virtual and public
  - No data members
- Review the sample code shapes.cpp in module folder.

Multiple Inheritance example

- Derived class inherits data and functions from all bases

```
class Phone
{ ...
  int number_;

  void call();
};
```

```
class Camera
{ ...
  int lens_;

  void shoot();
};
```

```
class SmartPhone : public Phone, public Camera
{ ...
  void sendPhoto();
};
```

Ambiguity

- Base classes may have members with the same name, use scope resolution to disambiguate where necessary

```
class Phone {
  ...
  int memoryAvailable();
};
```

```
class Camera {
  ...
  int memoryAvailable();
};
```

```
class SmartPhone :
      public Phone, public Camera {
  ...
};
```

```
SmartPhone p;

n = p.memoryAvailable();
n = p.Phone ::memoryAvailable();
n = p.Camera::memoryAvailable();
```

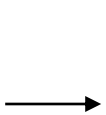
error, ambiguous →

ok →

Multiply inherited data

- A class may indirectly derive twice from a base class, it will inherit two copies of the data members

SmartPhone
gets two copies



```
class Device {
...
  int _id;
};
```

```
class Phone : public Device
{ ...
};
```

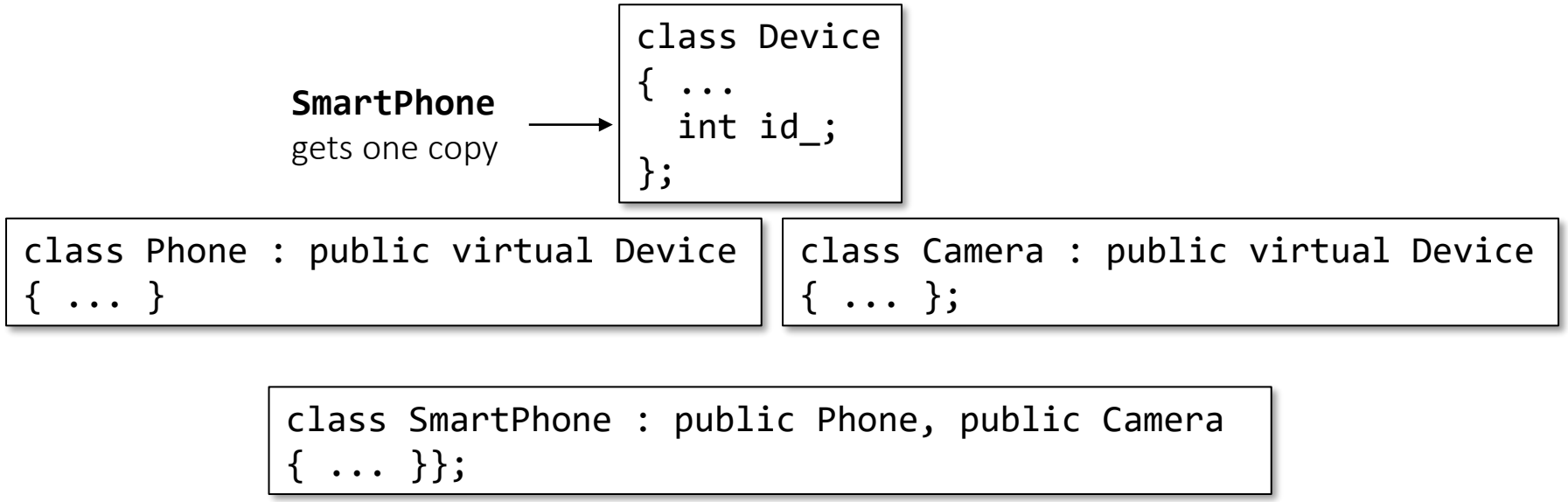
```
class Camera : public Device
{ ...
};
```

```
class SmartPhone : public Phone, public Camera
{ ...
};
```

error, ambiguous

Virtual Inheritance

- *Virtual base classes* give one copy of multiply-inherited data
- Review the sample code device.cpp in module folder.



Summary

- A class may have multiple base classes
  - may introduce ambiguities
- Multiple inheritance may be virtual
  - one copy of shared-base-class data inherited