C and C++
V

Spring 2014

Carola Wenk

C++ Inheritance

```
class A {
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};

class B : public A {
    // x is public, y is protected
    // z is not accessible from B
};

class C : protected A {
    // x,y are protected
    // z is not accessible from C
};

class D : private A {
    // x, y are private
    // z is not accessible from D
};
```

"extends"

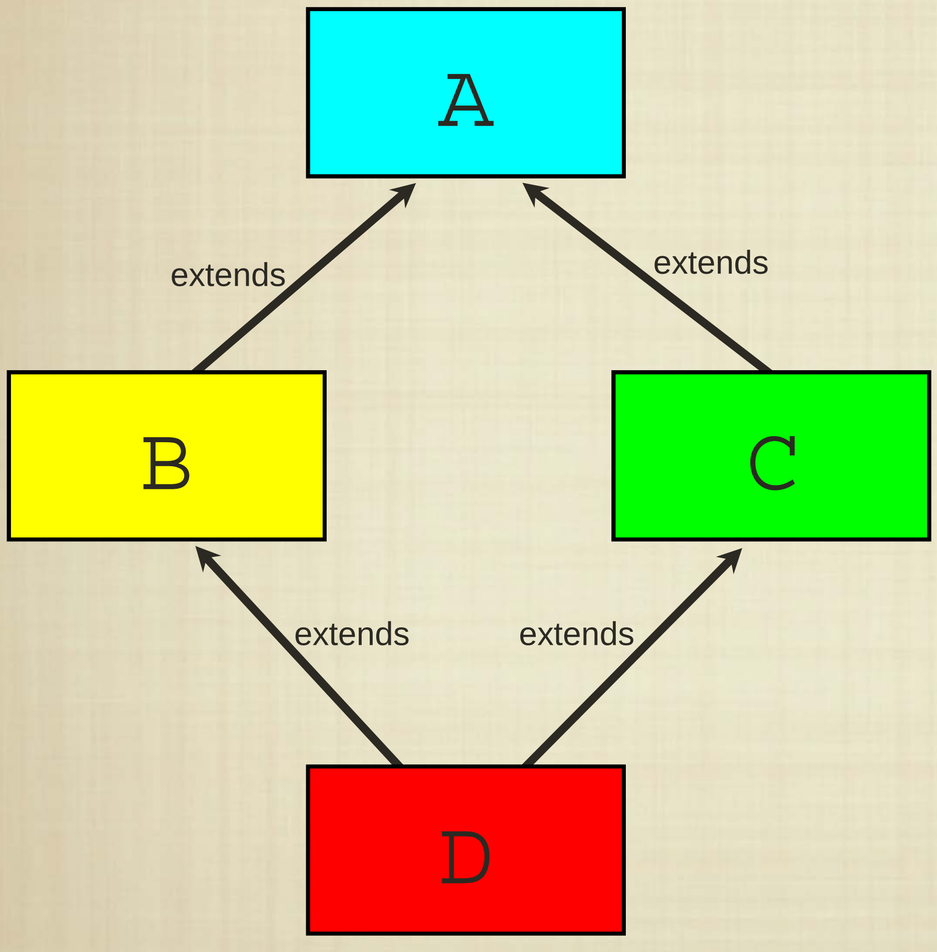
All members keep their original access specifications

```
class A {
    int x;
  public:
    int y;
};

class B {
    int a;
  public:
    int b;
};

class C : public A, public B {

};

class D : public A {
    int a;
};

class E : public B, public D {

};
```

Which a is it?

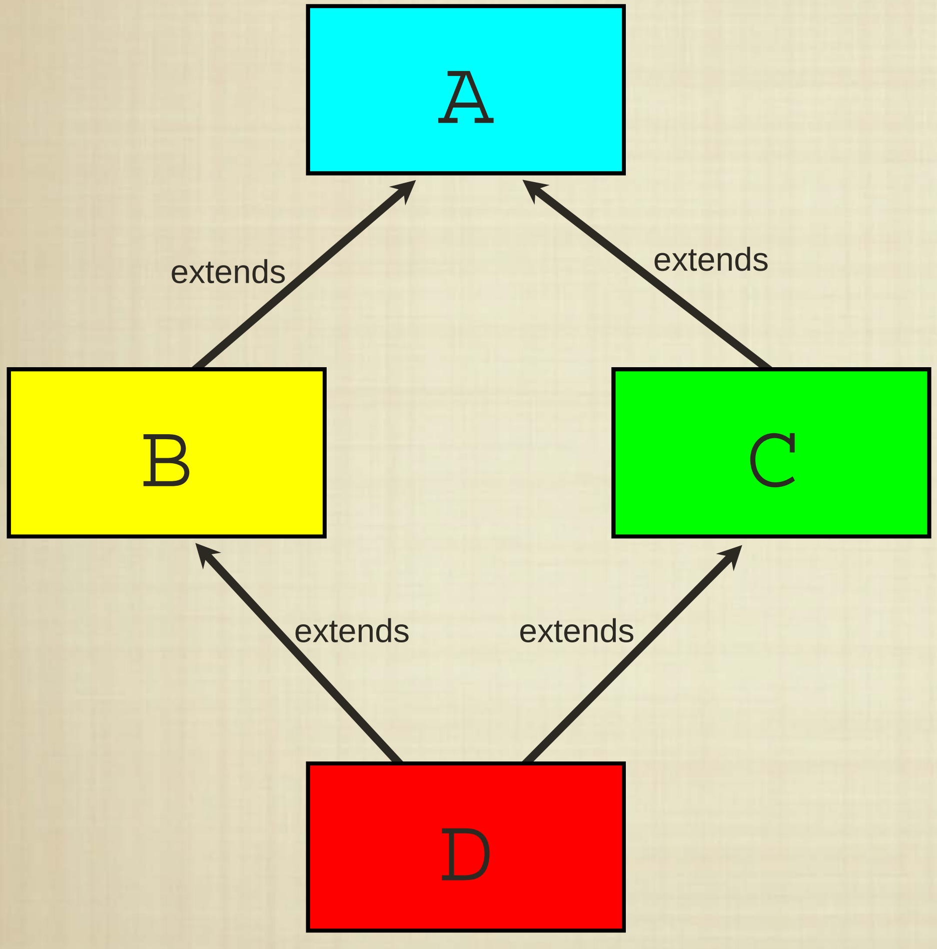
Inheritance in C++ is more flexible (and dangerous) than in Java. Overriding rules are the same, and C++ can access superclasses using a scope prefix.

Deadly Diamond of Death



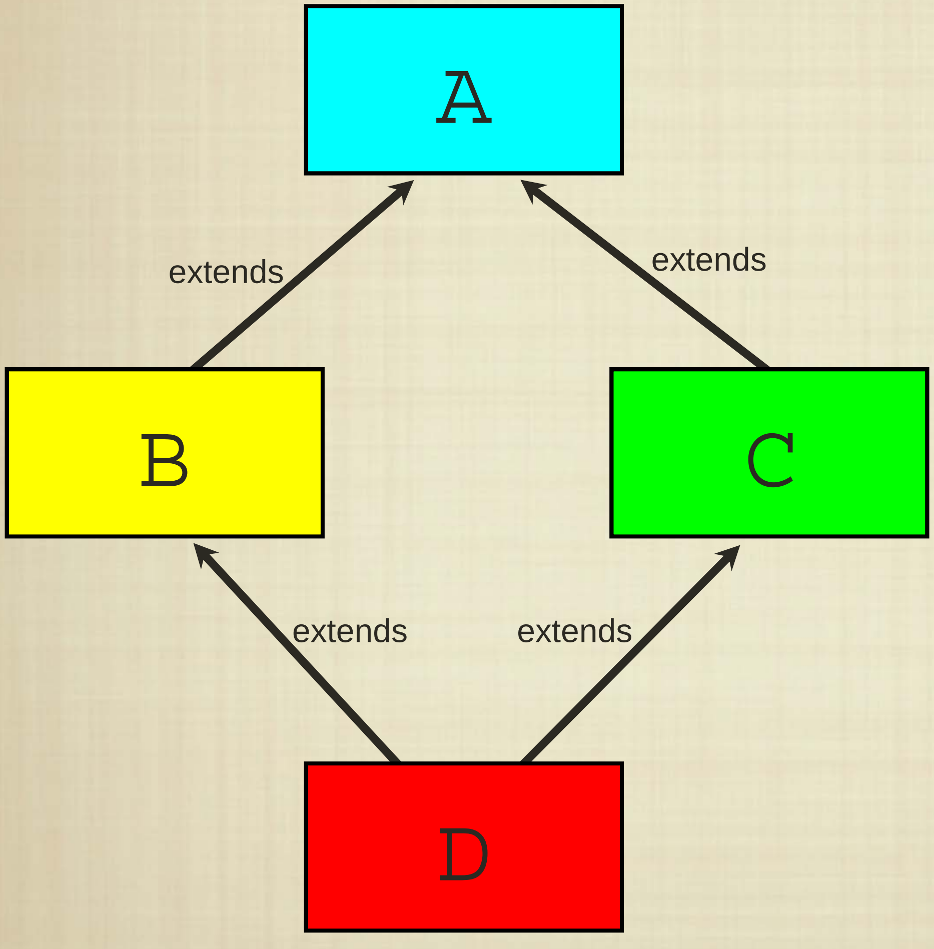
```
class A {
  public:
    int foo(){
      return 0;
    }
};

class B: public A {
  public:
    int foo(){
      return 1;
    }
};

class C: public A {
  public:
    int foo(){
      return 2;
    }
};

class D: public B, C {
  public:
    int bar(){
      return foo();
    }
};
```

Deadly Diamond of Death



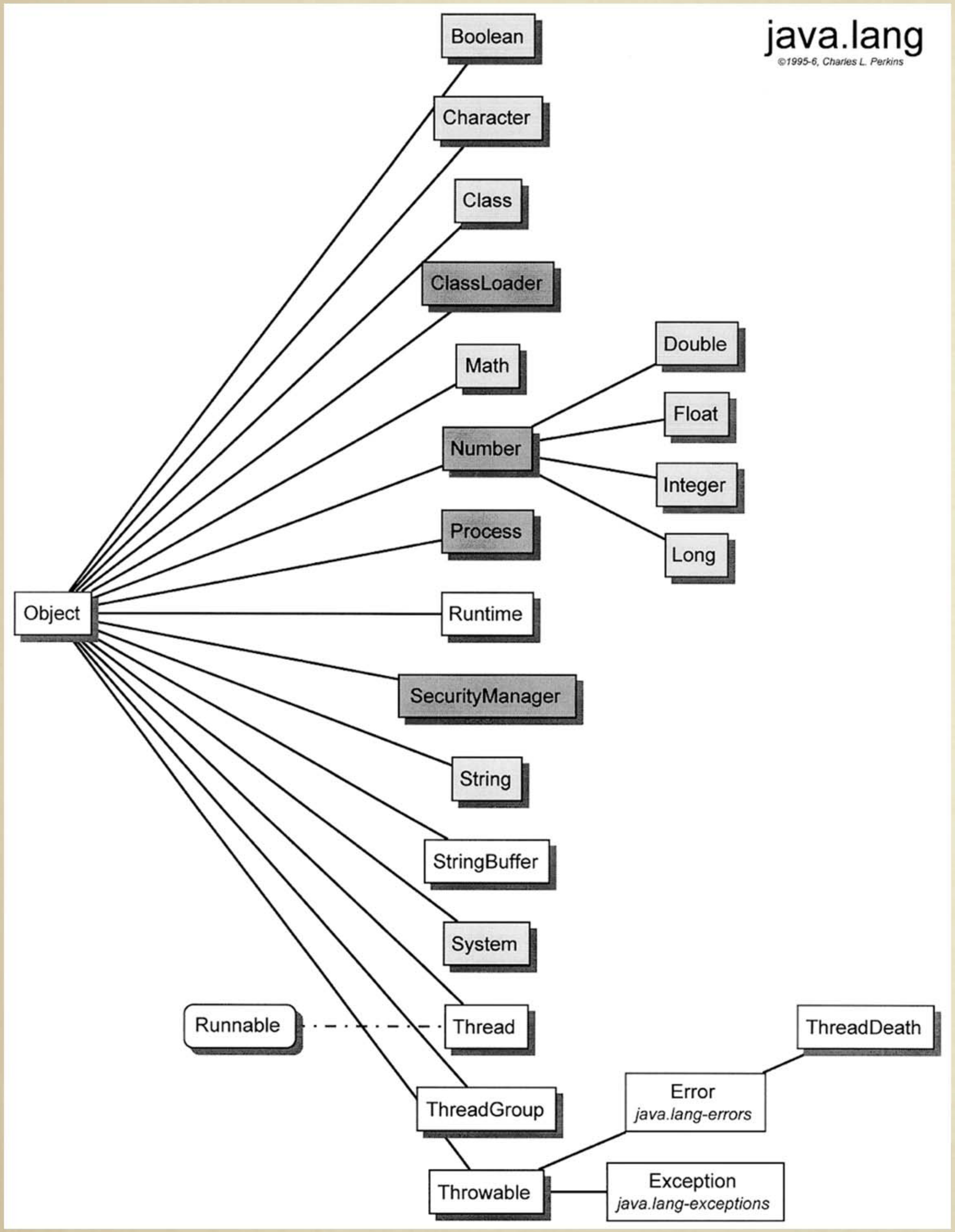
```
class A {
  public:
    int foo(){
      return 0;
    }
};

class B: public A {
  public:
    int foo(){
      return 1;
    }
};

class C: public A {
  public:
    int foo(){
      return 2;
    }
};

class D: public B, C {
  public:
    int bar(){
      return B::foo();
    }
};
```

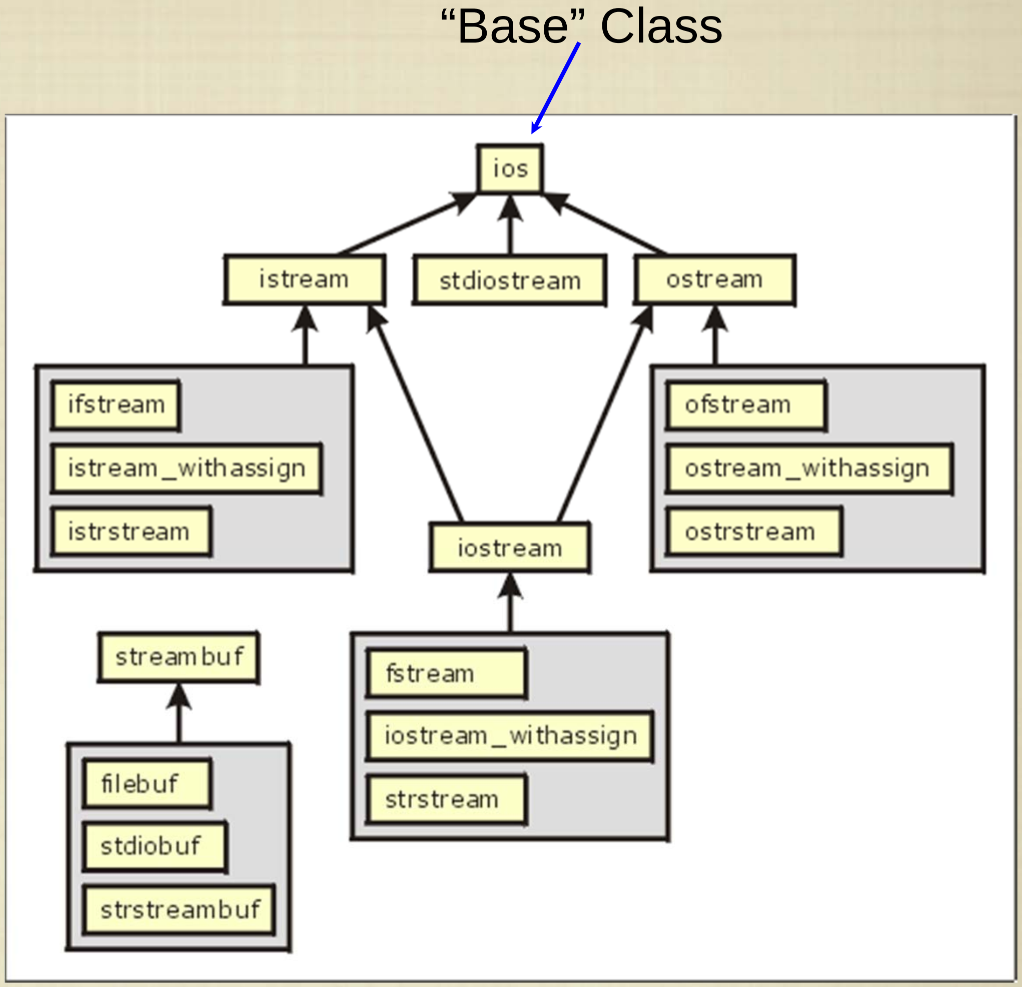
Deadly Diamond of Death



```
class A {
  public:
    int foo(){
      return 0;
    }
};

class B: public A {
  public:
    int foo(){
      return 1;
    }
};

class C: public A {
  public:
    int foo(){
      return 2;
    }
};

class D: public B, C {
  public:
    int bar(){
      return C::foo();
    }
};
```

C++ Standard Library





Standard "stream" classes

Interfaces

- C++ does not have interfaces. But because it has multiple inheritance, we can get interface-like behavior by defining an "*abstract class*" using (pure) `virtual` functions.

```
class MouseActions {
  public:
     virtual void mousePressed(...)=0;
     virtual void mouseReleased(...)=0;
     ...
};
```

No implementation. Method with this header has to be implemented by derived class.

```
class Foo : private MouseActions {

...

};
```

An "abstract class" is the analog of a Java interface or a Java abstract class. If a class inherits from an abstract class, it must implement all of its functionality.

C++ Templates

- Like Java, C++ provides libraries that contain implementations of various “collection” data structures.

- Many of these utilize `template` classes and functions, which mirror Java generics.

```
template <class T>
class Stack {
  int top;
  T* storage;
  public:
     Stack();
     Stack(int capacity);
     void push(T x);
     T pop();
};
```

```
template <class T>
Stack<T>::Stack() { ... }

template <class T>
Stack<T>::Stack(int capacity) { ... }

template <class T>
void Stack<T>::push(T x) { ... }

template <class T>
T Stack<T>::pop() { ... }
```