Objects/Inheritance Wrapup

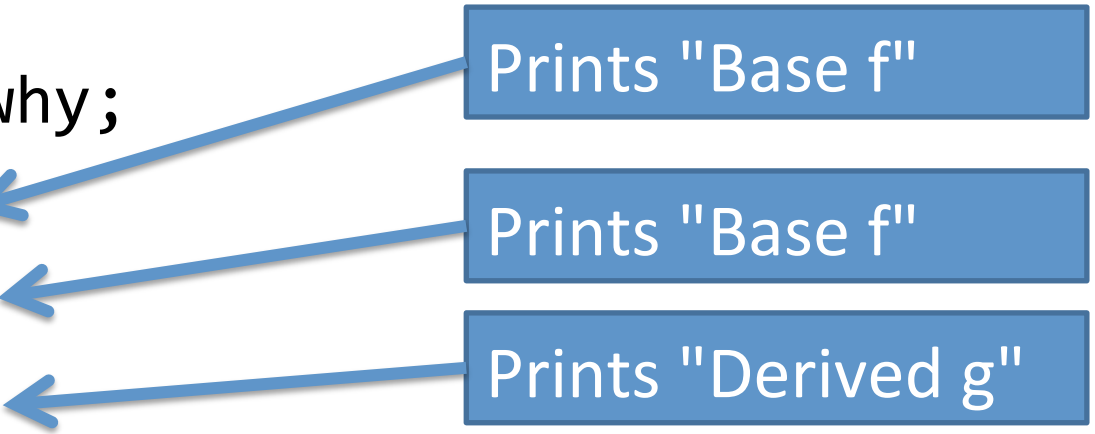
- **Class**: description of a data type that can contain fields (variables) and methods (functions)
  - Think of a class as a template for creating objects.
- **Object**: a particular instance of a class.

```
class point { … };
point p1, p2;
```

point is the class.
p1 and p2 are objects of the point class.

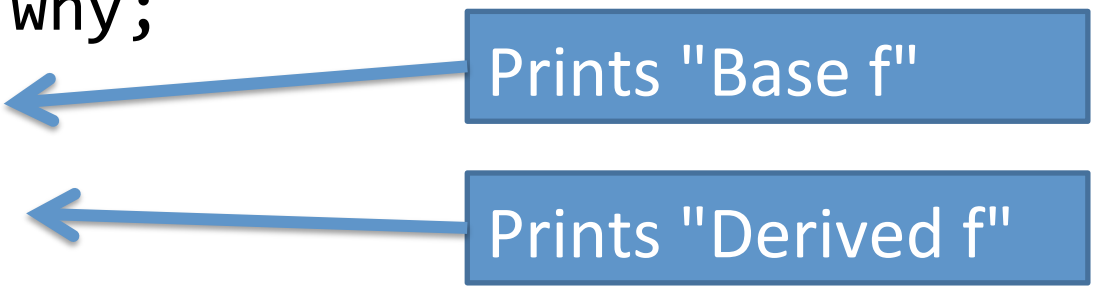
- When a class is a particular kind of another class, use **inheritance**.

```
class X { void f(); };
class Y : public X { void g(); };
void X::f() { cout << "Base f"; }
void Y::g() { cout << "Derived g"; }

X ex; Y why;
ex.f();
why.f();
why.g();
```



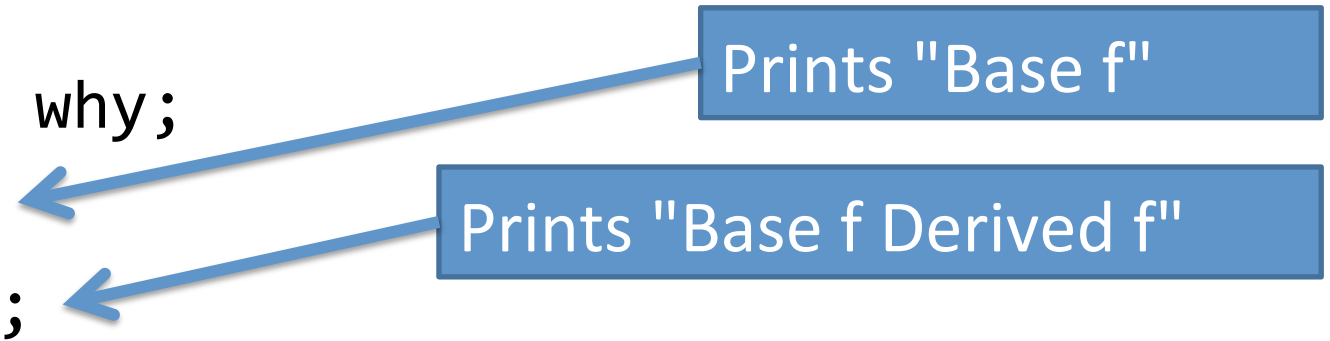
- A derived class is allowed to **override** methods in the base class.

```
class X { void f(); };
class Y : public X { void f(); };
void X::f() { cout << "Base f"; }
void Y::f() { cout << "Derived f"; }

X ex; Y why;
ex.f();
why.f();
```



- If a derived class overrides a method, the overridden method code can still call the base class version of the method if needed.

```
class X { void f(); };
class Y : public X { void f(); };
void X::f() { cout << "Base f"; }
void Y::f() { X::f(); cout << "Derived f"; }

X ex; Y why;
ex.f();
why.f();
```



- Sometimes a class needs access to "itself" as a stand-alone object:

```
class X { void f(); };

void g(const X & ex) { … }

void X::f() {
  // how can I call g on myself?
}
```

- Every object has a special variable called `this` that is available to be used inside any method in the class.
- `this` is always a pointer to the object itself.
- In other words, for a class X, the data type of `this` is X*.

- Sometimes a class needs access to "itself" as a stand-alone object:

```
class X { void f(); };

void g(const X & ex) { … }

void X::f() {
  g(*this);
}
```

- We know that the keyword const declares that a function will not change an argument:

```
void g(const vector<int> & vec) { … }
```

- We know that the keyword const declares that a function will not change an argument:

```
void g(const vector<int> & vec) { … }
```

- This const keyword can also be used with a class's methods to declare that the method will not change any of the object's fields.

```
class point {
  public:
    int get_x();
    int get_y();
  private:
    int x, y;
};
int point::get_x() {
  return x;
}
int point::get_y() {
  return y;
}
```

```
class point {
  public:
    int get_x() const;
    int get_y() const;
  private:
    int x, y;
};
int point::get_x() const {
  return x;
}
int point::get_y() const {
  return y;
}
```