Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

Week 5-2

ViSLAB
High-Performance Visualization, Communication, Computing

Member function Overriding and Polymorphism

Member function Overriding and Polymorphism

- Overloading:

Member function Overriding and Polymorphism

- Overloading:
  - same function name

Member function Overriding and Polymorphism

- Overloading:
  - same function name
  - different arguments

Member function Overriding and Polymorphism

- Overloading:
  - same function name
  - different arguments
- Overriding: (via Inheritance)

Member function Overriding and Polymorphism

- Overloading:
  - same function name
  - different arguments
- Overriding: (via Inheritance)
  - same function name

Member function Overriding and Polymorphism

- Overloading:
  - same function name
  - different arguments
- Overriding: (via Inheritance)
  - same function name
  - same arguments

Member function Overriding and Polymorphism

- Overloading:
  - same function name
  - different arguments
- Overriding: (via Inheritance)
  - same function name
  - same arguments
  - different implementation

Function Overriding

- a function defined in a super class
- the same function (name/args) can be "overridden" in the subclass

```
   class MyClass {
public:
   void func1(int a) { cout << a << endl;};
   virtual void func2(char* s){cout << s << endl;};
};

class NewClass : public MyClass {
public:
   void func2(char* s) {cout << "NewClass:func2 " << s << endl;};
};

int main() {
   NewClass obj;
   obj.func1(123);

   obj.func2("I'm NewClass");

   return 0;
}
```

Function Overriding

- overridden function
  - the same function name,
  - the same return type,
  - the same args.

Function Overriding

- If you think a function might be overridden by the subclass(es)
  - make it "virtual" function.
  - you could make all member function "virtual"....but

```
class MyClass {
public:
   void func1(int a) { cout << a << endl;};
   virtual void func2(char* s){cout << s << endl;};
};

class NewClass : public MyClass {
public:
   void func2(char* s) {cout << "NewClass:func2 " << s << endl;};
};

int main() {
   NewClass obj;
   obj.func1(123);

   obj.func2("I'm NewClass");

   return 0;
}
```

add "virtual"

same prototype

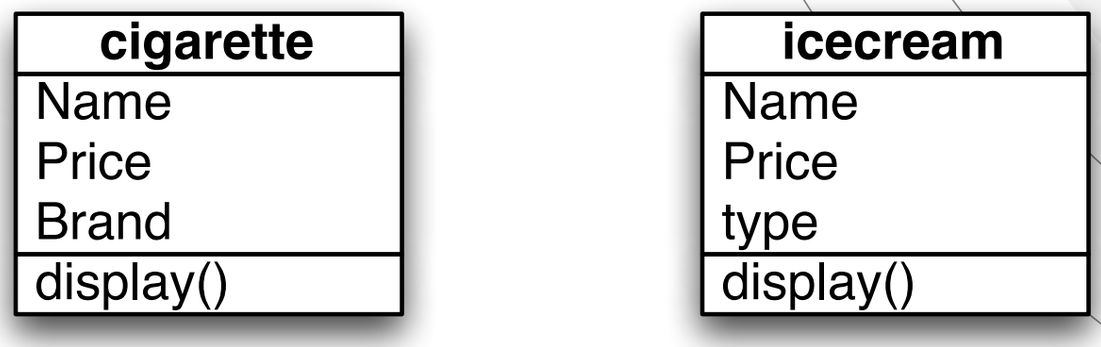
don't need to worry about

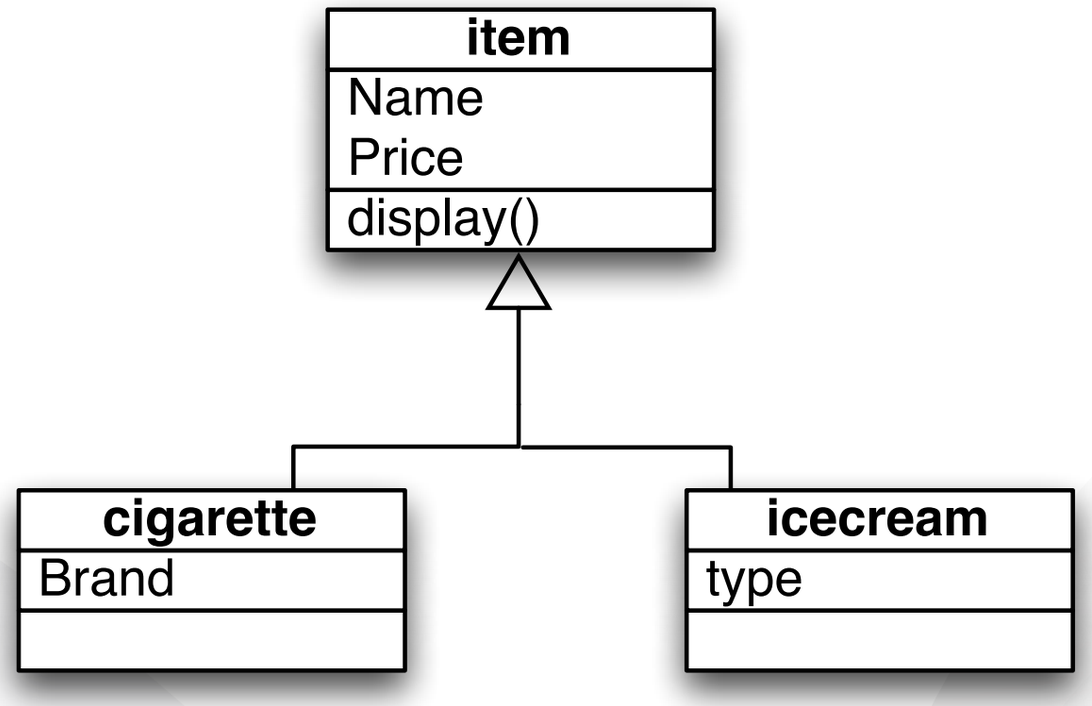
You can still call o.r. function

```
void MyClass::func2(char* s) {
   cout << s << endl;

}

void NewClass::func2(char* s) {
   cout << "NewClass:func2" << endl;
   MyClass::func2(s);
}
```

scope resolution operator

Overriding and Polymorphism





```
#include <iostream>
#include <string>

using namespace std;

class Item {
protected:
    string Name;
    float price;
public:
    Item(string name, float price):Name(name),price(price){};
    virtual void display();
};

void Item::display() {
    cout << Name << ": " << price << endl;
}

class Cigarette : public Item {
private:
    string Brand;
public:
    Cigarette(string name, float price, string brand):Item(name, price), Brand(brand){};
    void display();
};

void Cigarette::display() {
    Item::display();
    cout << "Brand: " << Brand << endl;
}

class Icecream : public Item {
private:
    int type;
public:
    Icecream(string name, float price, int type):Item(name, price),type(type){};
    void display();
};

void Icecream::display() {
    Item::display();
    cout << "type: " << type << endl;
}

int main() {
    Cigarette* cig = new Cigarette("A", 5.00f, "mine");
    Icecream* ice = new Icecream("B", 4.00f, 5);

    cig->display();
    ice->display();

    return 0;
}
```

```
    #include <iostream>
#include <string>

using namespace std;

class Item {
protected:
    string Name;
    float price;
public:
    Item(string name, float
price):Name(name),price(price){};
    virtual void display() = 0;
};


class Cigarette : public Item {
private:
    string Brand;
public:
    Cigarette(string name, float price, string
brand):Item(name, price), Brand(brand){};
    void display();
};

void Cigarette::display() {
    cout << Name << ": " << price;
    cout << "Brand: " << Brand << endl;
}
```

```
class Icecream : public Item {
private:
    int type;
public:
    Icecream(string name, float price, int
type):Item(name, price),type(type){};
    void display();
};

void Icecream::display() {
    cout << Name << ": " << price;
    cout << "type: " << type << endl;
}

int main() {
    Cigarette* cig = new Cigarette("A", 5.00f,
"mine");
    Icecream* ice = new Icecream("B", 4.00f, 5);

    cig->display();
    ice->display();

    return 0;
}
```

```cpp
#include <iostream>
#include <string>

using namespace std;

class Item {
protected:
    string Name;
    float price;
public:
    Item(string name, float price):Name(name),price(price){};
    virtual void display();
};

void Item::display() {
    cout << Name << ": " << price << endl;
}

class Cigarette : public Item {
private:
    string Brand;
public:
    Cigarette(string name, float price, string brand):Item(name, price), Brand(brand){};
};
```

```cpp
#include <iostream>
#include <string>

using namespace std;

class Item {
protected:
    string Name;
    float price;
public:
    Item(string name, float price):Name(name),price(price){};
    virtual void display() = 0;
};

class Cigarette : public Item {
private:
    string Brand;
public:
    Cigarette(string name, float price, string brand):Item(name, price), Brand(brand){};
    void display();
};

void Cigarette::display() {
    cout << Name << ": " << price;
    cout << "Brand: " << Brand << endl;
}
```