



Data Structures with C++ : CS189

Lecture 2-2: Polymorphism

Inheritance

Recap

- When we have many classes and want to reduce duplicated code, we use inheritance
 - I have a Dog and a Cat, but their code could be elsewhere
- When we have many classes, but we want to write code that doesn't care what they are, we use polymorphism
 - If I have a Dog object and an Animal pointer, it is still a dog

Polymorphism

Recap

- "virtual" makes method calls start at the leaf instead of where their pointer is
- "pure virtual" makes a class abstract so objects can't be made from it
 - Syntax of "= 0"
- Inheritance + virtual = Polymorphism

Why virtual?

```
class Animal {};
```

```
class Dog : public Animal  
{  
    bool IsAlive();  
};
```

```
Dog *X = new Dog;  
X->IsAlive(); //Yay
```

```
Animal *Z = X;  
Z->IsAlive(); // NO
```

- I have a Dog and an Animal pointer to it
- I want to call IsAlive, but Animal is empty
- Basic inheritance says "You have an Animal pointer so you can call Animal methods. IsAlive does not exist"
- Polymorphism says "You have an Animal pointer and with no virtual methods to tell me to look elsewhere, IsAlive does not exist"
 - Virtual exists specifically for this issue

Baseclass Clutter

```
class Animal {
```

```
virtual bool IsAlive() = 0;
```

```
virtual bool IsCute() = 0;
```

```
virtual bool IsHungry() = 0;
```

```
virtual bool IsHappy() = 0;
```

```
virtual void Func() = 0;
```

```
·
```

```
·
```

```
·
```

```
};
```

- So it follows that to use polymorphism, every method anywhere in the inheritance tree must be pure virtual in the baseclass
- A good baseclass that covers a large amount of other classes could have 100 pure virtual methods
 - Everything any kind of Animal can do

Interface

```
class IBurnable {  
    virtual void  
    CatchFire() = 0;  
    virtual void  
    Extinguish() = 0;  
};
```

```
class Car :  
    public Vehicle,  
    public IBurnable  
{};
```

- An interface is nothing but a way to organize pure virtual methods
- Instead of Animal having 100 methods, it could implement 20 interfaces that each have 5 methods
 - Code isn't changing, just its location
- Interface names start with capital I
- A class can implement as many as they want
- An interface can only have pure virtual methods

Using Interfaces

- To use an interface, you have to ask the object if they have that interface
- The `dynamic_cast` command tries to change the pointer you have in to the interface
- Careful looking this up online.
"__interface" and "^" aren't standard C++
 - They're Microsoft specific

```
class IBurnable{  
public:  
    virtual void CatchFire() = 0;  
    virtual void Extinguish() = 0;  
};  
class Car : public Vehicle,  
    public IBurnable {};
```

```
Vehicle *tCar = new Car;  
IBurnable *tAsBurnable =  
dynamic_cast<IBurnable*>(tCar);  
if( tAsBurnable )  
    tAsBurnable->CatchFire();
```

- If tCar isn't nullptr, then yes it burns and you can use burn code
- Without the interface, the Vehicle gets 100 methods
- You even get polymorphic bonuses
 - "I don't care what kind of object you are or who your base class is. Do you burn?"

An Alternative

```
class Animal {  
    int type;// enum  
    bool IsCow();  
    bool IsDog();  
    int GetType();  
};  
  
void Func(Animal *ptr) {  
    if( ptr->IsCow() )  
        Cow *now = (Cow*)ptr;  
}
```

- Store the name of the leaf class in a base class property
 - Most likely an Enum
- Give the baseclass methods that check against that name
- Calling code uses old C-cast
 - We know it is safe because we just checked it
- This is another code style thing

Recap

- Inheritance is ISA
 - Dog ISA Mammal
- Polymorphism is inheritance plus virtual methods
- A property is HASA
 - Dog HASA Tail
- An interface is CAN
 - A car CAN burn
 - IBurnable
 - An animal CAN eat
 - IDigestion

Pause

Now is the day to ask for help if you don't *completely* understand any of the listed topics.

- Nothing else new this week
- This is the end of the C++ Advanced Review
- If you have trouble with any of the below topics, you're gonna have a bad time
 - Objects
 - Pointers
 - Dynamic memory allocation
 - Constructor / Destructor
 - Inheritance
 - Polymorphism



End

Just about every data structure we make will hold polymorphic data