Inheritance, Abstraction, and Polymorphism

PRESENTED BY: JOSH

Inheritance

- You already know how to make classes
- In C++ you can also make subclasses (called derived classes)
- Derived classes inherit all of the non-private features of their base class
- This is called inheritance

Using Inheritance

- Let's say that we have a class called Animals:

```
class  Animals {
        protected:
                int height, weight;
                string sound;
        public:
                string getSong(){ return "House of the Rising Sun"; }
                int Animals::getHeight(){ return height; }
                int Animals::getWeight(){ return weight; }
                string Animals::getSound(){ return sound; }
                void Animals::setAnimals(int h, int w, string s){
                        height = h;
                        weight = w;
                        sound = s;
                }
                Animals::Animals(){}
                ~Animals(){ cout << "Don't let me be Misunderstood" << endl;}
};
```

Using Inheritance cont'd

- And we wanted to make a class Bird, a type of animal :

```
class Bird : public Animals{
        string getSong(); // overwrites parent function in private
        public :
                string Bird::speak(){
                        return "Polly want a cracker";
                }
};
```

Using Inheritance Example

- Using the class definition from the previous two slides and assuming everything necessary was included (string, iostream …) what would happen if :

```
Bird bird;
Animals eric;
bird.setAnimals(1, 1, "quack");
eric.setAnimals(1, 1, “hi");
cout << bird.getSound() << endl;
cout << bird.speak() << endl;
cout << eric.getSound() << endl;
cout << eric.getSong() << endl;
cout << eric.speak() << endl;
cout << bird.getSong() << endl;
```

Virtual Functions

- With inheritance, you might not always want a function in your base class to return something.
- Or reserve implementation for the derived class.
- This is where virtual functions come in.
- Similar to abstract functions in Java.

Virtual function syntax

- You have two choices. You can either say:

```
virtual int func(){ return 1;}
```

- Or, you can say:

```
virtual int func() = 0;
```

- The second way is known as a pure virtual function

Abstract classes

- An abstract base class is a class with at least one pure virtual function.
- They cannot make objects, but can be a base class for another class which can.
- Additionally, pointers to one are still valid.

Abstract classes cont'd

```
class Animal {
        string noise;
        double weight;
        public:
                virtual string getSize(double weight) = 0;
                string getNoise(){ return noise;}
                Animal(string s = "", double w = 0){ noise = s; weight = w;}
                ~Animal(){}
};
class Lion : public Animal{
        public:
                Lion(string s = "", int w = 0) : Animal(s, w) {}
                string getSize(double weight){  return (weight < 1000) ? "small" : "large";
};
class Ant : public Animal{
        public:
                Ant(string s = "", int w = 0) : Animal(s, w) {}
                string getSize(double weight){  return (weight < .1 ) ? "small" : "large";
};
```

Abstract Classes example

- From the code on the previous page, would the following produce an error?

```
Animal* animal1 = new Ant("skittle", 1);

Animal* animal2 = new Lion("Roar", 1);
```

Polymorphism

- By using the virtual function and the abstract classes, we are able to take one function and depending on the type called with the function, produce a different result. This is an example of polymorphism.

- From the previous two slides, what would these two lines do:

```
cout << animal1->getSize() << endl;

cout << animal2->getSize() << endl;
```

//NOTE: When calling a function for a pointer, use "->" instead of "."