Object Oriented programming and software engineering – Lab 10

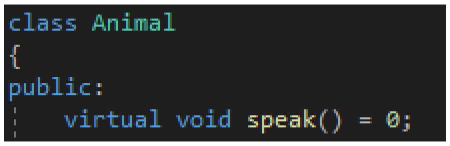
Adam Korytowski – 2025

1. Abstract classes

An abstract class is a class that cannot be instantiated on its own. It's used as a base class and typically serves as a blueprint for other classes.

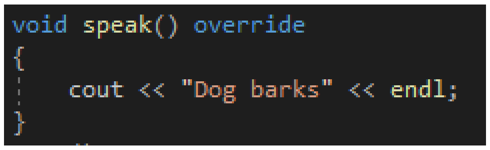
A class becomes abstract when it has at least one pure virtual function.

What's a Pure Virtual Function? – It's a virtual function that doesn't have a definition in the base class — it's meant to be overridden in derived classes.



```cpp
class Animal
{
public:
    virtual void speak() = 0;
```

Then in child class *Dog*:



```cpp
void speak() override
{
    cout << "Dog barks" << endl;
}
```

Why Use Abstract Classes?

- Define a common interface for all derived classes.
- Enforce that derived classes must implement certain methods.

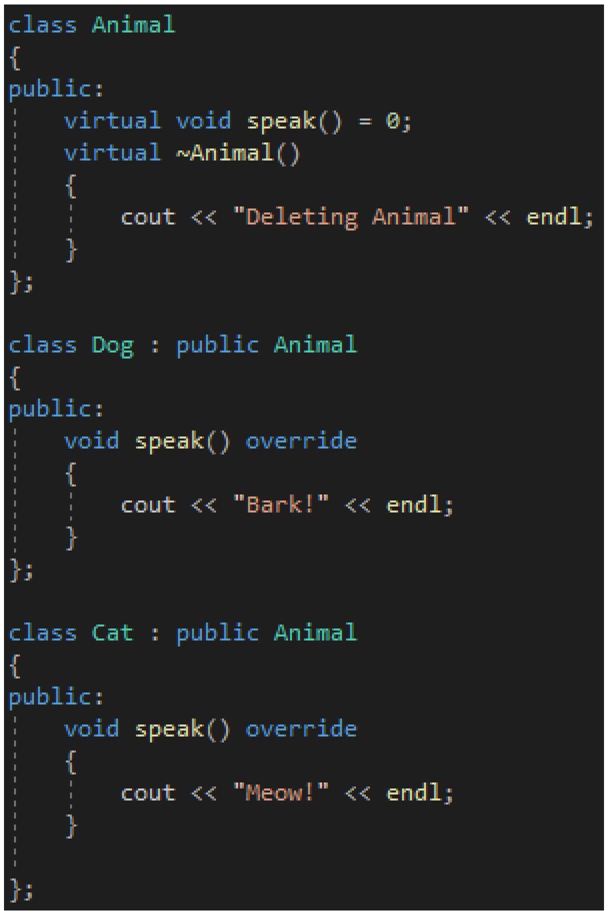
2. Polymorphism

Polymorphism – in C++, it allows one interface (like a base class pointer or reference) to represent multiple underlying types (derived classes).

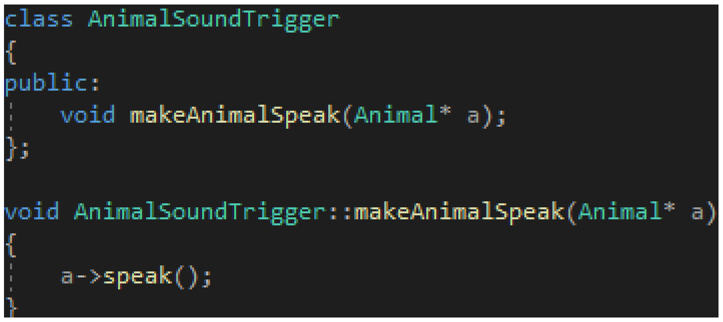
In simpler terms:

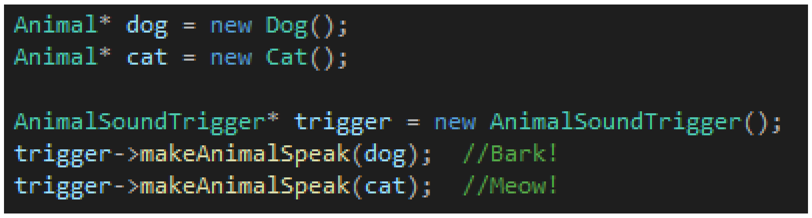
You can use a base class pointer to call methods on objects of derived classes, and the correct method gets called automatically at runtime.

Example:


```cpp
class Animal
{
public:
    virtual void speak() = 0;
    virtual ~Animal()
    {
        cout << "Deleting Animal" << endl;
    }
};

class Dog : public Animal
{
public:
    void speak() override
    {
        cout << "Bark!" << endl;
    }
};

class Cat : public Animal
{
public:
    void speak() override
    {
        cout << "Meow!" << endl;
    }

};
```

Then if we implement a function/method that calls the virtual function, the output will depend on the type of our deriving class.


```cpp
class AnimalSoundTrigger
{
public:
    void makeAnimalSpeak(Animal* a);
};

void AnimalSoundTrigger::makeAnimalSpeak(Animal* a)
{
    a->speak();
}
```


```cpp
Animal* dog = new Dog();
Animal* cat = new Cat();

AnimalSoundTrigger* trigger = new AnimalSoundTrigger();
trigger->makeAnimalSpeak(dog);  //Bark!
trigger->makeAnimalSpeak(cat);  //Meow!
```

3. Tasks (3 pts each)

- Modify one of your base classes to be an abstract class, modify your code so that objects of the abstract class are never created, only objects of your child classes
- Implement polymorphism – create a function that, as an argument, accepts pointer to your abstract class and calls virtual function from this class (the function should be overridden in child classes), present how the function works – the output should be different depending on type of the pointer (different output for pointers of different child classes)
- Rearrange your code so that each class is separated into .h and .cpp files, plus one separate file for the main function.