

Swinburne University Of Technology

Faculty of Science, Engineering and Technology

LABORATORY COVER SHEET

| | |
|------------------------------|------------------------------|
| Subject Code: | COS30008 |
| Subject Title: | Data Structures and Patterns |
| Lab number and title: | 2 part 2, Inheritance |
| Author: | Carmen Chai |

Class Inheritance in C++

The goal of this laboratory session is to learn how to use class inheritance and explore more about the base class and the derived class.

Part 1 – Constructors and Destructors**Project Setup**

First, set up your project.

Create a main.cpp and an Animal class (.h and .cpp). The Animal class will be our base class for this lab example.

Next, create another class (in our example we'll use Cat, feel free to name it as any other animal you like :D).

Let this class be a derived class of Animal. Do this by adding `#include "Animal.h"` to the top of Cat.h file, and adding

: `public Animal` to the first line of your Cat class container.

For example:

```
class Cat : public Animal
{
};
```

Base class specifications

The base class will have no variables. But it will have a constructor, destructor and 2 member functions in public.

The constructor will print out that it is the base constructor, while the destructor will print out that it is the base destructor. (i.e. "Animal constructor", "Animal destructor") so you can tell when it is being called.

The two functions are void functions that print out the name (sayName) and the call (makeCall) of the animal respectively.

Be sure to include what you need (such as iostream), and add the namespace you may require as well (std).

Derived class specifications

Be sure to include "Animal.h" at the top as it is the base class.

The derived class will have its own constructor and destructor. Change the text output to differentiate them. (i.e. "Cat constructor", "Cat destructor")

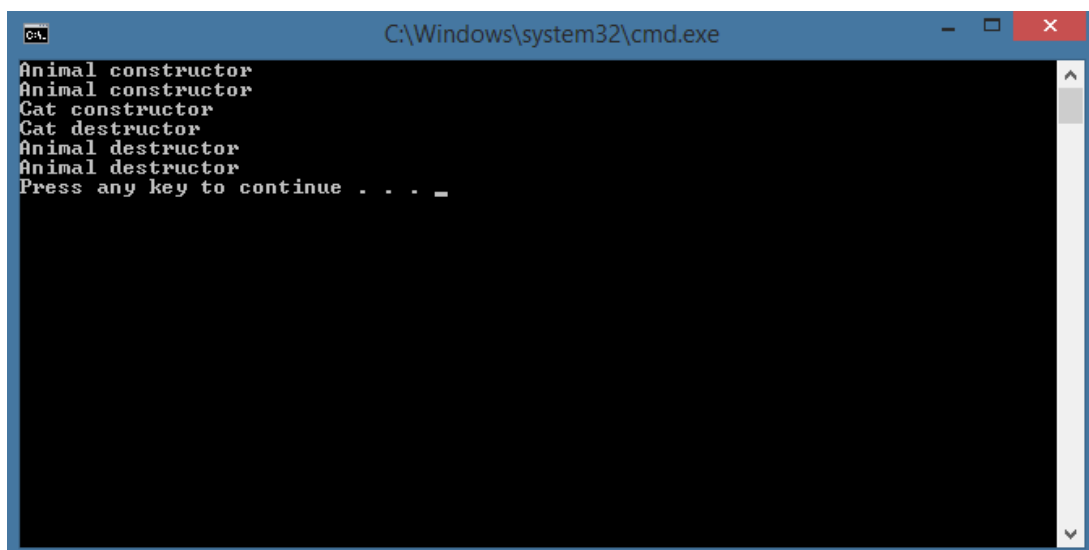
Main.cpp specifications

Since we are using both base class and derived class in this example, be sure to have both "Animal.h" and "Cat.h" included at the top.

The target of this exercise is to see how the base class and derived class are related. In your main() function, simply create an Animal object and a Cat object and run your program. Remove either the Animal object or the Cat object and observe the difference.

You should see that in order for the derived object to start working, it needs to create a base class object first, that is why you still see an Animal object being constructed and destructed without actually having an animal object.

Example Output Screenshot:



```
C:\Windows\system32\cmd.exe
Animal constructor
Animal constructor
Cat constructor
Cat destructor
Animal destructor
Animal destructor
Press any key to continue . . . _
```

Part 2 – Polymorphism and Virtual Functions

In this part of the lab, we will explore polymorphism and virtual functions.

Project Setup

Create a new project. We only require main.cpp for this one.

We'll be using iostream and namespace std for this project as well so add those in.

Create an Animal class in main.cpp as the base class. Create two other classes that inherit from Animal in main.cpp as well, in this example we will be using Cat and Bird.

Base class specifications:

It has one *protected* member, an int variable that stores the number of legs "numberOfLegs".

If the variable were made private, the derived class would not be able to access it directly. However, making it public would mean anything outside of the class would be able to access and alter its value. Therefore, we use protected.

It has two public functions.

- i. A virtual void function that prints out how the animal moves.
- ii. A pure virtual function for printing out how the animal calls.

A virtual function means that if the derived class has overridden this, the derived class' version of the function will be used instead. If it is not overridden, the base class' version will be used.

A virtual function becomes pure when it has no body. Instead of {} it has a =0; A pure virtual function has no implementation of its own so the derived classes must override it. Additionally, when a class has a pure virtual function, it becomes an abstract class.

Derived classes specifications:

In the move function, specify the number of legs the animal has to numberOfLegs. For example, a Cat would have 4 legs while a bird would have 2. Use cout << to print the values to the console when this function is called.

In the call function, simply put a sentence that makes sense of the animal. For example, a cat meows while a bird chirps.

The content of the main function:

```
int main() {
    Cat cat;
    Bird bird;

    Animal *animal1 = &cat;
    Animal *animal2 = &bird;

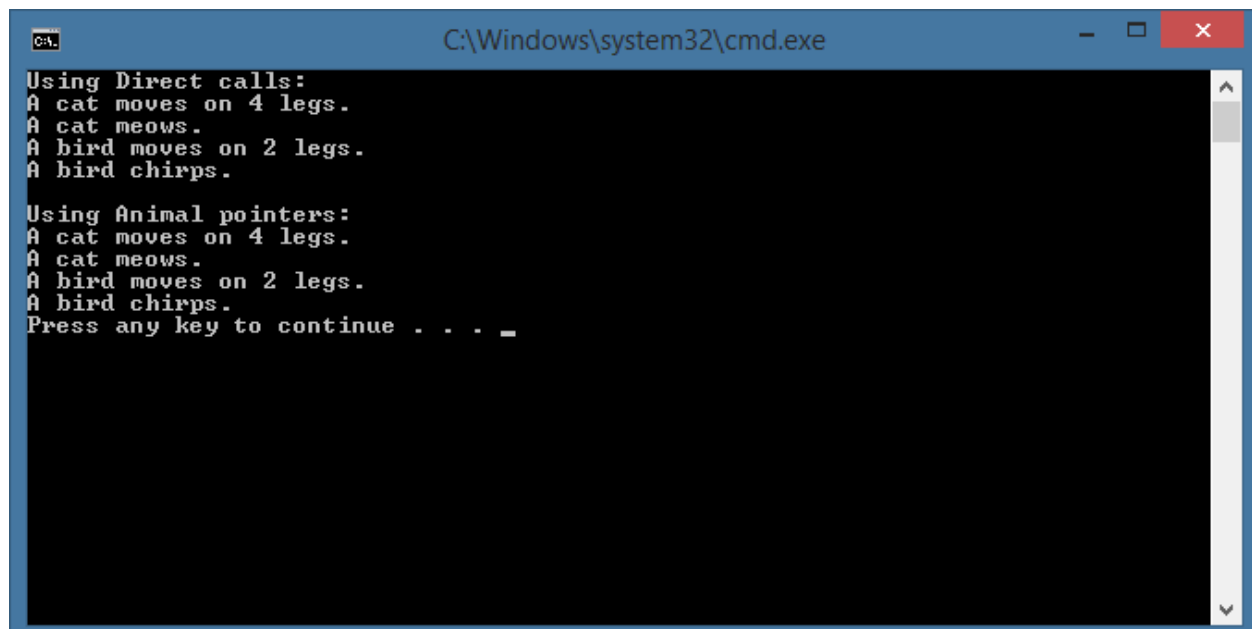
    cout << "Using Direct calls: " << endl;

    cat.move();
    cat.call();
    bird.move();
    bird.call();

    cout << "\nUsing Animal pointers: " << endl;

    animal1->move();
    animal1->call();
    animal2->move();
    animal2->call();

}
```

Example Output Screenshot:

```
C:\Windows\system32\cmd.exe

Using Direct calls:
A cat moves on 4 legs.
A cat meows.
A bird moves on 2 legs.
A bird chirps.

Using Animal pointers:
A cat moves on 4 legs.
A cat meows.
A bird moves on 2 legs.
A bird chirps.
Press any key to continue . . . _
```