

## II.6. Object-Oriented Programming Concepts

While “Structured Programming” introduces structure by organizing functions into a logical framework, Object-Oriented Programming (OOP) introduces structure by grouping both functions and data in a logical framework. For example, in C++:

```
class Cat
{
    unsigned int age;
    float weight;
    void Meow();
};
```

This class defines the attributes and behavior of cats in general, but doesn’t tell us anything about our particular cat (named Frisky). An “object” is a particular instance of a class. Frisky (an object) is a cat (a class).

Since Frisky is an object of the type cat, Frisky takes up memory in the computer and we can assign values to the data.

```
Frisky.age = 5;
Frisky.weight = 7;
```

All Object Oriented Programming languages are built upon three pillars: encapsulation, inheritance, and polymorphism.

### Encapsulation

It means an object can be treated as a “black-box”. We can use it without understanding the details of how it works. (E.g. We know Frisky, and any other cat, can Meow(), but don’t know how it does this.) As well, the internal state of the object can be hidden from the outside world. This is referred to as “data-hiding”. (E.g., we could make “age” inaccessible to non-cat objects.)

### Inheritance

It means we can create a new class by basing it on an existing one. All of the functions and data from the parent super-class are inherited by the child sub-class. This avoids duplicating code that is common between classes. Inheritance is used in two circumstances:

(i) Defining a more specific class of something. E.g., Siamese is a type of cat:

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

SiameseCat automatically has all of the functions and data from Cat.

- (ii) To “adapt” an existing class for an unanticipated use or to define a communication interface between objects. For example, the set of Cat classes may have been developed independently of a set of “Animal\_sounds” classes. We can use inheritance to make a new class SiameseCat\_sound that inherits both from SiameseCat and Animal\_sounds. SiameseCat\_sound objects have all of the functionality of SiameseCat but can be used wherever Animal\_sounds objects can be used. In the example below, the Meow() function from SiameseCat is used to perform the getSound() function from Animal\_sounds.

```
class Animal_sounds
{
    void getSound();
};
```

```
class SiameseCat_sound : public SiameseCat, virtual public
    Animal_sounds
{
    void getSound()      { Meow(); }
};
```

## Polymorphism

It means that an object of a sub-class can be treated the same as an object of the super-class, even though the sub-class may have redefined the behavior from the super-class. That is, a function in the parent class may result in one action, but the equivalent function in the sub-class may do something completely different. With polymorphism, the correct function will get called according to the class of the object. E.g.,

```
Frisky.Meow();
```

will normally result in:

```
Meow.
```

But if Frisky is a SiameseCat and SiameseCats make a different sound, then Frisky will make a different sound when Frisky.Meow() is called.

## A note on “Interfaces”

An “interface” is a class that has only functions and no data, with the functions left undefined. The functions in the interface class don’t provide any behavior, but instead define protocols for object communication. Objects of classes derived from the interface class define behavior for each function and thus can be communicated with using the defined functions. If the Animal\_sounds class above does not have any behavior of its own, then it would actually be considered an interface class. Note that SiameseCat\_sound defines a behavior for the getSound() function.

## Preparation code

```
#include <iostream>

using namespace std;

class Cat
{
public:
    unsigned int age;
    float weight;
    void Meow();
};

void Cat::Meow()
{
    cout << "Meow." << endl;
}

int main()
{
    Cat Frisky;
    Frisky.age = 5;
    Frisky.weight = 7.;
    Frisky.Meow();
    cout << "Frisky is a cat who is ";
    cout << Frisky.age << " years old." << endl;
    Frisky.Meow();
    return 0;
}
```