Class Types

In this lesson, we will explore some of the properties of classes.

WE'LL COVER THE FOLLOWING

Definition

> Classes are data types encapsulating **attributes** and **methods**

C++ has several primitive classes such as integer, string, double, etc. Each type has its own methods that allow us to perform certain operations.

For example, the `length()` method can be used to calculate the length of a string.

In C++, we can create custom class types using the built-in types and defining our own methods.

Classes are quite similar to structs. For the purpose of this course, we will stick to classes.

Here is an example of a class:

```
class Account{
```

```
    public:
        Account(double b);
        void deposit(double amt);
        void withdraw(double amt);
        double getBalance() const;
    private:
        double balance;
};
```

Attributes

Attributes are variables defined in the class body. The variable, `balance`, is an attribute or *data member* since it stores data relevant to the class.

Methods

The functions of a class are known as *methods*. Methods usually perform an operation using the attributes of the class. We will discuss the `public` flag in a bit.

Objects

> An object is an instance of a class.

Classes help us create objects that have the same properties.

> 💡 Objects are concrete examples of a class that exist at run time.

Object instantiation

Creating a class object is the same as creating a string or integer. We must specify the class name, followed by the name of the variable.

```
Account account(100.0);
```

Here, we:

- Create an object, `account`, of type `Account`.
- Initialize the object.

- When it is created, it is known to the compiler.

One may wonder, "*what will the object be capable of*? The answer lies in the class's public methods and its attributes.

Accessing class members

All public methods can be accessed from outside the class. This means that for the `Account` class above, `deposit()`, `withdraw()`, etc. can be accessed directly for each object.

Each object will also have a `balance` attribute but it cannot be accessed directly since it is private. Only the methods of the class can use this data member.

The `.` operator can be used to access the members of a class. In the case of object pointers, `->` is used instead.

```
Account acc(200.0);
Account* accPtr = new Account(200.0);

acc.balance = 500.0;          // ERROR; balance is private
acc.deposit(200.0);

accPtr->balance = 500.0; // ERROR; balance is private
accPtr->deposit(200.0);
```

> **Note**: Based on the access rights, the compiler decides whether access to a class member is possible or not.

`static` attributes

> A static, or class attribute, is an attribute *shared* by all instances of a class.

Usually, each object generates its own copy of the class's attributes, i.e., the attributes of one object are independent of the attributes of another.

This is not true for a *static* attribute. Only one instance of the attribute is

present and shared among all the objects.

Think of it as a global class variable.

A static attribute can also be accessed without a class instance because it is part of the class, not the individual object. To access it, we can use the scope resolution operator, `::`.

> Static class attributes can only be initialized outside the class.

```
class Account{
    ....
    static int deposits;
};
int Account::deposits = 0;
```

The next lesson highlights the purpose of constructors in classes.