Constructors

In this lesson, we'll learn how to create an instance of a class.

WE'LL COVER THE FOLLOWING

In the previous lesson, we learned what classes and objects are.

> **Constructors** are special methods for the instantiation of an object in the class.

It is declared inside the class but can be defined outside as well. The constructor:

- has the *exact* same name as the class.
- does not have a return type.

Default constructors

Let's revisit the `Account` class from the previous lesson:

```
class Account{
    public:
        Account(); // Constructor
        void deposit(double amt);
        void withdraw(double amt);
        double getBalance() const;
    private:
        double balance;
};
```

Here, `Account()` is the constructor and has the same name as the class.

It can be used to create an instance of the class in the following two ways:

```
Account acc;
Account *acc = new Account;
```

This is very similar to how we declare basic type variables. Well, it turns out that all the basic types also call their class constructors!

Notice, the constructor above has no parameters. This constructor is known as the **default constructor**.

When the default constructor is called, the attributes of the object are initialized to their default values.

We can specify this value inside the implementation of the constructor:

```
...
public:
  Account(){
    balance = 5000;
  }
...
```

Now, whenever an object of the `Account` class is created using the default constructor, the default value of the `balance` attribute will be `5000`. If a default value is not specified, the compiler will assign a value itself.

Calling the default constructor invokes the constructors of the base classes being used in this class as well.

> **Note**: The compiler needs the default constructor to automatically create instances.

Parameterized constructor

The parameterized constructor allows us to set the values of an object's attributes by passing them as arguments in the constructor.

attributes by passing them as arguments in the constructor.

As the name suggests, the constructor has parameters.
Let's suppose we want to define the `balance` of an `Account` object when it is created. We will need a parameterized constructor.

```
class Account{
    public:
        Account(); // Constructor
        Account(double bal); // Parameterized constructor
    ...
```

What would be inside the parameterized constructor?

Let's have a look:

```
...
public:
  Account(double bal){
    balance = bal;
  }
...
```

Now, the value for `balance` can be specified when an instance is created.

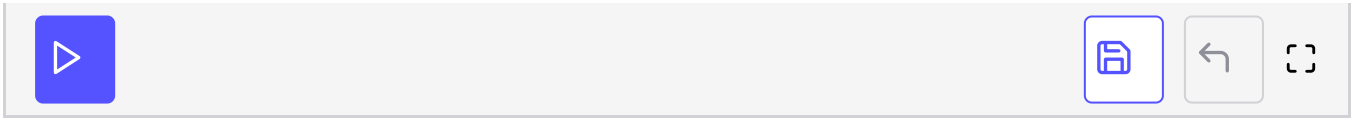
```
Account acc(100.0);
Account* acc = new Account(100.0);
```

Example

Here's a working example that incorporates both constructors:

```
#include <iostream>

class Account{
public:
  Account(){
    std::cout << "Account: this: "  << this << std::endl;
  };
  Account(double){
    std::cout << "Account(double): this: "  << this << std::endl;
  }
};

int main(){

   std::cout << std::endl;
```

```
    Account account1;
    Account* account2 = new Account;


    std::cout << std::endl;

    Account account3(100.0);
    Account* account4 = new Account(100.0);

    std::cout << std::endl;

    Account* accountArray = new Account[10];

    std::cout << std::endl;

}
```



- On lines 17 and 18, we can see the default constructor creating `Account` objects on the stack and heap, respectively.
- The parameterized constructor simply takes in a double as an argument. Since we don't do anything with it, there is no need to give the parameter a name.
- On lines 22 and 23, the `double`'s arguments cause the parameterized constructor to be called instead of the default constructor.
- We can also create a dynamic array of type `Account`, as done in line 27. This statement calls the default constructor `10` times because the size of the array is `10`.

We'll explore more constructors in the next lesson.