Virtual Methods

In this lesson, we'll discuss virtual methods in detail.

WE'LL COVER THE FOLLOWING

Why do we use virtual methods?

- Virtual methods are used to adjust the behavior of an object while keeping its interface stable.
- In order to override a method, it must be declared `virtual`.

For documentation purposes, the overriding method will also be declared as `virtual`.

Example

```
struct Account{
  virtual void withdraw(double amt){ balance -= amt;

}
...

struct BankAccount: Account{
  virtual void withdraw(double amt){
    if ((balance - amt) > 0.0) balance -= amt;
...
```

The dynamic type of the object determines which version of a virtual method will be called. To apply virtuality, a pointer or a reference is needed.

```
BankAccount bankAccount(100.0);

Account* aPtr = &bankAccount;
aPtr->withdraw(50.0);

Account& aRef = bankAccount;
aRef.withdraw(50.0);
```

Polymorphic:

- The method is selected at run-time.
- This is often called dynamic or late binding.

Rules:

- Constructor must not be virtual.
- Virtual methods do not have to be overridden.
- Methods declared as virtual stay virtual in the hierarchy.
- The parameters in the overridden method must be identical to the parameters in the virtual method.
- The specifier virtual is required to obtain polymorphic behavior.
- Private methods can be overridden in the base class.

> There's a difference between overriding and overloading.

Virtual destructors

When destructing an object via a pointer or a reference to a base class, the destructors must be virtual.

```
Account* aptr;

Account* aptr = new BankAccount(100.0);
delete aptr;

Account& aRef = BankAccount(200.0);
```

In the next lesson, we'll look at an example of virtual methods.