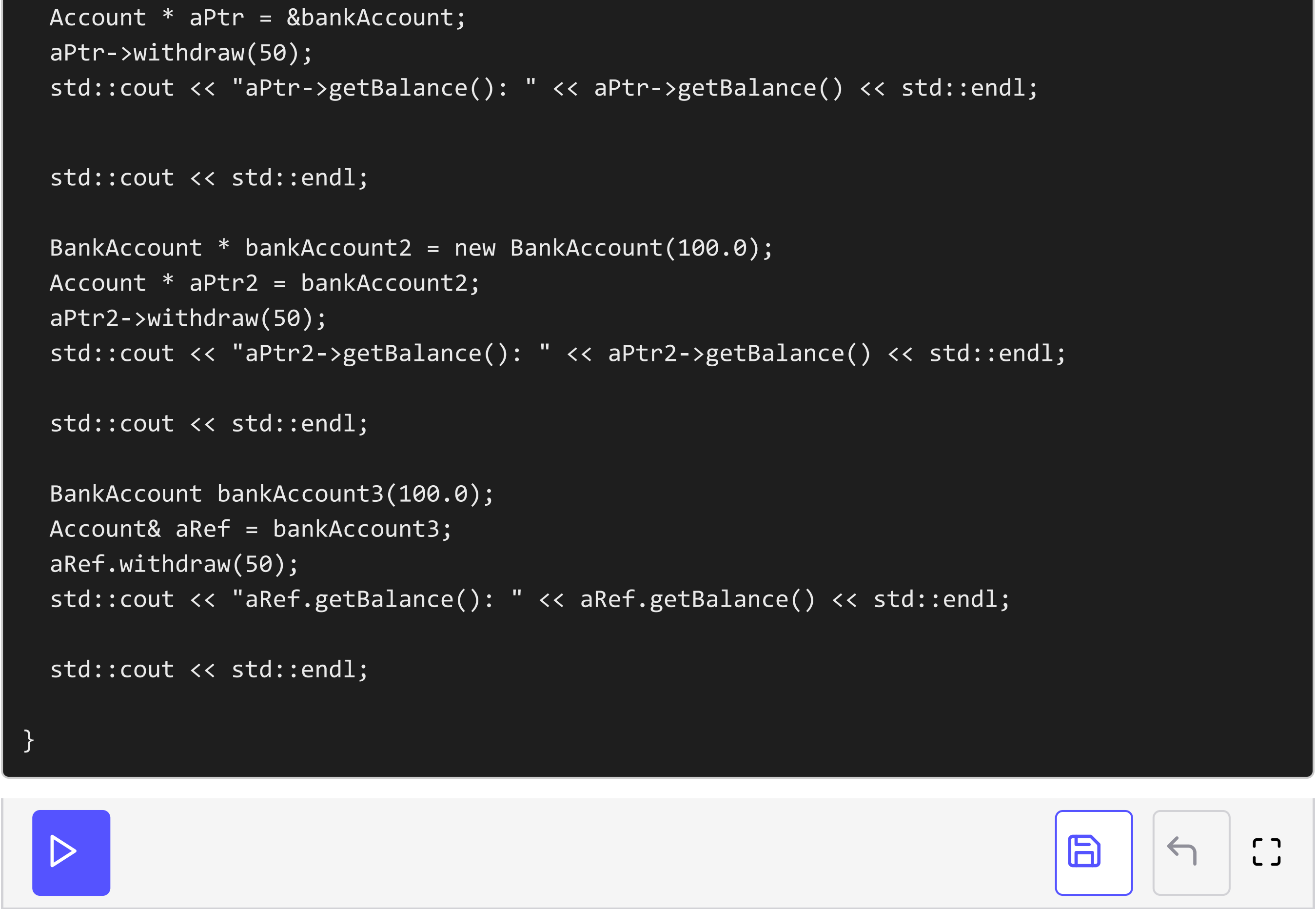
- Example

In this lesson, we'll look at an example of a virtual method.

WE'LL COVER THE FOLLOWING

Example: Virtual methods

```
#include <iostream>

class Account{
public:
  Account(double amt): balance(amt){}

  virtual void withdraw(double amt){
    balance -= amt;
  }

  double getBalance() const {
    return balance;
  }

protected:
  double balance;
};


class BankAccount: public Account{
public:

  BankAccount(double amt): Account(amt){}

  virtual void withdraw(double amt){
    if ((balance - amt) > 0.0) balance -= amt;
  }

};

int main(){

  std::cout << std::endl;

  BankAccount bankAccount(100.0);
```



```
  Account * aPtr = &bankAccount;
  aPtr->withdraw(50);
  std::cout << "aPtr->getBalance(): " << aPtr->getBalance() << std::endl;


  std::cout << std::endl;

  BankAccount * bankAccount2 = new BankAccount(100.0);
  Account * aPtr2 = bankAccount2;
  aPtr2->withdraw(50);
  std::cout << "aPtr2->getBalance(): " << aPtr2->getBalance() << std::endl;

  std::cout << std::endl;

  BankAccount bankAccount3(100.0);
  Account& aRef = bankAccount3;
  aRef.withdraw(50);
  std::cout << "aRef.getBalance(): " << aRef.getBalance() << std::endl;

  std::cout << std::endl;

}
```

Explanation

- In the example above, we have implemented two classes, `Account` and `BankAccount`.
- The `BankAccount` class publicly inherits from `Account` class.
- We have implemented a `virtual` function in both of these classes named `withdraw` in lines 7 and 25.
- We can access the `withdraw` function of the `BankAccount` class using three approaches by calling a reference to the derived class from the base class in line 36.
- By assigning a pointer of the base class to the newly created object using `new` keyword of the derived class in line 43.
- By using the reference of the base class, we can point to the derived class object. By using the .operator, we can access the derived class method in line 50.

In the next lesson, we'll work through an exercise to better understand virtual methods.