- Examples

In this lesson, we'll discuss the examples of multiple inheritance.

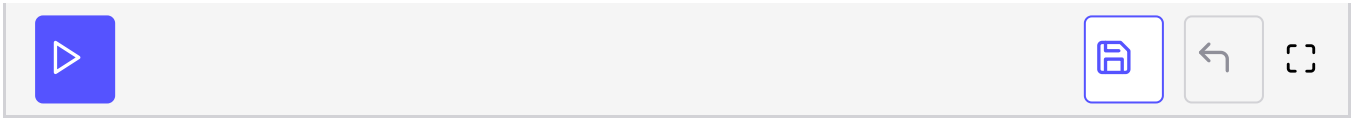
WE'LL COVER THE FOLLOWING

- Example 1: Multiple inheritance
  - Explanation
- Example 2: Virtual multiple inheritance
  - Explanation

Example 1: Multiple inheritance

```
#include <iostream>

class Account{
public:

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

  double getBalance() const {
    return amount;
  }

private:
  double amount;
};

class BankAccount: public Account{
public:
  BankAccount(double amt): Account(amt){}
};

class WireAccount: public Account{
public:
  WireAccount(double amt): Account(amt){}
};

class CheckingAccount: public BankAccount, public WireAccount{
public:
  CheckingAccount(double amt): BankAccount(amt), WireAccount(amt){}
};

int main(){
```

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


  CheckingAccount checkAccount(100.0);
  // checkAccount.getBalance();         // ERROR

  std::cout << "checkAccount.BankAccount::getBalance(): "  <<  checkAccount.BankAccount::getB
  std::cout << "checkAccount.WireAccount::getBalance(): "  <<  checkAccount.WireAccount::getB

  std::cout << std::endl;

}
```



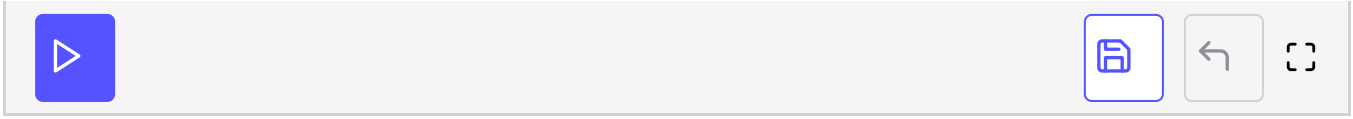
Explanation

- In the example above, we have created an `Account` class with a `getBalance` method.
- The `BankAccount` and `WireAccount` classes publically inherit from the `Account` class and have access to the `getBalance` method.
- The class `CheckingAccount` publicly inherits from `BankAccount` and `WireAccount` in line 26, and now has access to the `getBalance` methods of both classes.
- If we try to call the `getBalance` method using the instance of `CheckingAccount` class, it will give us an error, but if we call it with the name of the base class along with the scope operator `::` then it works fine.

Example 2: Virtual multiple inheritance

```
#include <iostream>

class Account{
public:

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

  double getBalance() const {
    return amount;
  }

private:
  double amount;
};
```

```
class BankAccount: virtual public Account{
public:
  BankAccount(double amt): Account(amt){}
};

class WireAccount: virtual public Account{
public:
  WireAccount(double amt): Account(amt){}
};

class CheckingAccount: public BankAccount, public WireAccount{
public:
  // CheckingAccount(double amt): BankAccount(amt), WireAccount(amt){}
  CheckingAccount(double amt): BankAccount(amt), WireAccount(amt),Account(amt){}
};

int main(){

  std::cout << std::endl;

  CheckingAccount checkAccount(100.0);
  std::cout << "checkAccount.getBalance(): " << checkAccount.getBalance() << std::endl;

  std::cout << "checkAccount.BankAccount::getBalance(): "  <<  checkAccount.BankAccount::getB
  std::cout << "checkAccount.WireAccount::getBalance(): "  <<  checkAccount.WireAccount::getB

  std::cout << std::endl;

}
```



Explanation

- In the example above, we have created all the classes in the same way as in example 1.
- The only thing that we have changed is virtually inheriting the `Account` class in the `BankAccount` and `WireAccount` classes.
- By inheriting these classes virtually, we can now access the `checkAccount.getBalance()` method of the `Account` class.

In the next chapter, we'll explore the world of templates.