- Examples

Here are a few examples of the different types of methods that we explored in the previous lesson.

WE'LL COVER THE FOLLOWING

Static methods

```
#include <iostream>

class Account{
public:
    Account(){
      ++deposits;
    }
    static int getDeposits(){
      return Account::deposits;
    }

private:
    static int deposits;
};

int Account::deposits= 0;

int main(){

  std::cout << std::endl;

  std::cout << "Account::getDeposits(): "  << Account::getDeposits() << std::endl;

  Account account1
  Account account1;
  Account account2;


  std::cout << "account1.getDeposits(): "  << account2.getDeposits() << std::endl;
  std::cout << "account2.getDeposits(): "  << account1.getDeposits() << std::endl;
  std::cout << "Account::getDeposits(): "  << Account::getDeposits() << std::endl;

  std::cout << std::endl;

}
```

Explanation

- The static attribute, `deposits`, is being initialized on line 8. Whenever the constructor is called, its value is incremented by `1`.
- On lines 22 and 29, we can see that the static `getDeposits()` method can be called without an instance of `Account`.
- After the creation of `account1` and `account2`, the value of `deposits` is incremented to `2`. We can check this by invoking `getDeposits()` on the objects, as done on lines 27 to 29, or by using the scope resolution operator with the class name.

`this` pointer

```
#include <iostream>

class Base{

public:

  Base& operator = (const Base& other){
    if (this == &other){
      std::cout << "self-assignment" << std::endl;
      return *this;
    }
    else{
      a = other.a;
      b = other.b;
      return *this;
    }
  }

  void newA(){
    int a{2011};
    std::cout << "this->a: " << this->a << std::endl;
    std::cout << "a: " << a << std::endl;
    std::cout << "b: " << b << std::endl;
    std::cout << "this->b: " << this->b << std::endl;
  }


private:
  int a{1998};
  int b{2014};

};

int main(){

  std::cout << std::endl;

  Base base;
  base.newA();

  std::cout << std::endl;

  Base& base2 = base;
  base = base2;

  std::cout << std::endl;

}
```

Explanation

- The `this` pointer is being used in the copy operation on lines 8, 10, and 15.
- To check whether the assignee and assigned are the same object, we can use `this`. In such a case, we can simply return the dereferenced value of the `this` reference to our object.
- The class contains two attributes, `a` and `b`.
- In the `newA()` method, there is a variable named `a`. We can differentiate between the variable and the attribute by using `this` to access the attribute, as done on line 21.
- Since there isn't a `b` variable in the method, `this->b` and `b` mean the same thing (line 24). This is because every member already has an implicit `this` pointer.

Constant methods

```
#include <iostream>
```

```
#include <iostream>

class Account{

public:
  double getBalance() const {
    return balance;
  }
  void addAmount(double amount){
    balance += amount;
  }
private:
  double balance{0.0};
};

int main(){

  std::cout << std::endl;

  Account readWriteAccount;
  readWriteAccount.addAmount(50.0);
  std::cout << "readWriteAccount.getBalance(): " << readWriteAccount.getBalance() << std::end

  const Account readAccount;
  std::cout << "readAccount.getBalance(): " << readAccount.getBalance() << std::endl;

  std::cout << std::endl;

}
```

Explanation

- The `const` method, `readAccount` is defined on line 23. It simply returns the value of `balance`.

- The `readWriteAccount` object on line 19 allows `balance` attribute to be modified through class methods. We can see an example of this on line 20.

- The `const` object, `readAccount`, can only invoke the `const readAccount` method. Using `addAmount` with it will throw an error. This is because `const` objects can only call other `const` methods.

`constexpr` methods

```
#include <iostream>

class Account{
public:
  constexpr Account(int amou): amount(amou){}
  constexpr double getAmount() const {
    return amount;
  }

  constexpr double getAccountFees() const {
    return 0.05 * getAmount();
  }
private:
  double amount;
};

int main(){

  std::cout << std::endl;

  constexpr Account accConst(15);
  constexpr double amouConst = accConst.getAmount();
  std::cout << "amouConst: " << amouConst << std::endl;
  std::cout << "accConst.getAccountFees(): " << accConst.getAccountFees() << std::endl;

  std::cout << std::endl;

  Account accDyn(15);
  double amouDyn = accDyn.getAmount();
  std::cout << "amouDyn: " << amouDyn << std::endl;
  std::cout << "accDyn.getAccountFees(): " << accDyn.getAccountFees() << std::endl;

  std::cout << std::endl;

}
```

Explanation

- The `constexpr` constructor, and the methods `getAccountFees` and `getAmount` will be evaluated at compile time.

- Since `constexpr` methods are implicitly `const`, we mention the `const` keyword in the definitions as well.

- Line 21 shows how the returned value of `getAmount()` can be stored in a `constexpr double`.

- As we can see in line 27, `constexpr` methods can also be called by non-`constexpr` objects.

In the next lesson, we will learn about **requests** and **suppressed** methods.