# Destructors

# Cleaning Up Objects

```
class Bank
{
  public:
    Bank();

  private:
    Account **accounts;
};
Bank::Bank()
{
  accounts = new Account*[Bank::MAX_ACCOUNTS];
  for (int i = 0; i < Bank::MAX_ACCOUNTS; i++)
    accounts[i] = NULL;
}
```

- Bank() dynamically allocates memory
- But it becomes inaccessible when the variable goes out of scope
- How do you free it?

# Destructors

```
class Bank
{
  public:
    Bank();
    ~Bank();

  private:
    Account **accounts;
};
```

```
Bank::~Bank()
{
  delete accounts;
}
```

- How do you free it? Add a destructor to do it for you
  - You won't call this function explicitly, it will be called when the object goes out of scope
  - This is where you handle all clean up of the object
  - You can also add these to
    - Perform additional tasks (e.g. logging)
    - Indicate that no cleaning needs to occur (e.g. no memory allocation takes place)
    - Indicate that cleaning should not be done (e.g. pointer is maintained elsewhere)

# When Constructors Get Called

- In addition to showing how the caller interfaces with the classes,
- The driver of this code shows when constructors / destructors take place
- The following examples will show
  - The driver
  - The output
  - Relevant code snippets

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Declaring a bank variable" << std::endl;
Bank bank;
```

Calls the default constructor

```
Declaring a bank variable
--Bank Constructor (0x7ffca3ddb9c0)
```

```
Bank::Bank()
{
  std::cout << "--Bank Constructor (" << this << ")\n";
}
```

Don't worry about *this* just yet

All you have to know is this is giving us the address of the object

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Creating a new bank account" << std::endl;
int accountId = bank.NewAccount("Fred");
```

```
Creating a new bank account
------Money(0) Constructor (0x11cf0b0)
----Account Constructor (0x11cf0a0)
```

```
int Bank::NewAccount(std::string name)
{
  accounts[i] = new Account(i, name);
  return accounts[i]->id;
}
Account::Account(unsigned int identifier, std::string owner)
{
  std::cout << "----Account Constructor (" << this << ")\n";
}
Money::Money()
{
  std::cout << "------Money(0) Constructor (" << this << ")\n";
  amt = 0;
}
```

Calls Account Constructor

Immediately calls default Money Constructor

This is because It has a Money Member variable

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Declaring a money variable: check1" << std::endl;
Money check1(1000); //$10.00
std::cout << "Check 1 is ";
PrintDollars(check1);
std::cout << std::endl;
```

Calls Money Constructor

```
Declaring a money variable: check1
------Money(1) Constructor (0x7ffca3ddb950)
Check 1 is $10.0
```

```
Money::Money(int amount)
{
  std::cout << "------Money(1) Constructor (" << this << ")\n";
}
```

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Declaring a money variable: check2" << std::endl;
Money check2(500, 50); //$500.50
std::cout << "Check 2 is ";
PrintDollars(check2);
std::cout << std::endl;
```

Calls Money Constructor

```
Declaring a money variable: check2
------Money(2) Constructor (0x7ffca3ddb960)
Check 2 is $500.50
```

```
Money::Money(int dollars, int cents)
{
  std::cout << "------Money(2) Constructor (" << this << ")\n";
}
```

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Declaring a money variable: sumChecks" << std::endl;
Money sumChecks = check1 + check2; //$510.00
std::cout << "Check sum is ";
PrintDollars(sumChecks);
std::cout << std::endl;
```

Defers Constructor due to assignment

```
Declaring a money variable: sumChecks
------Money(1) Constructor (0x7ffca3ddb970)
Check sum is $510.50
```

```
Money Money::operator+(const Money &money) const
{
  Money ret(amt + money.amt);
  return ret;
}
Money::Money(int amount)
{
  std::cout << "------Money(1) Constructor (" << this << ")\n";
}
```

Constructor gets called here instead

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Updating sumChecks to include check3" << std::endl;
sumChecks += check3; //$910.75
std::cout << "Check sum is ";
PrintDollars(sumChecks);
std::cout << std::endl;
```

Updates existing object

```
Updating sumChecks to include check3
------Money(1) Constructor (0x7ffca3ddb990)
------Money Destructor (0x7ffca3ddb990)
Check sum is $514.75
```

```
Money Money::operator+=(const Money &money)
{
  amt += money.Amount();
  return amt;
}
Money::Money(int amount)
{
  std::cout << "------Money(1) Constructor (" << this << ")\n";
}
Money::~Money()
{
  std::cout << "------Money Destructor (" << this << ")\n";
}
```

A copy of sumChecks is created on return

But immediately goes out of scope because it's not used

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Depositing sumChecks into the bank" << std::endl;
bank.Deposit(accountId, sumChecks);
```

```
Depositing sumChecks into the bank
------Money(1) Constructor (0x7ffca3ddb910)
------Money(1) Constructor (0x7ffca3ddb920)
------Money Destructor (0x7ffca3ddb920)
------Money Destructor (0x7ffca3ddb910)
```

```
int Bank::Deposit(int id, const Money &money)
{
  accounts[i]->amountSaved += money.Amount();
  return 0;
}
Money::Money(int amount)
{
  std::cout << "------Money(1) Constructor (" << this << ")\n";
  amt = amount;
}
Money Money::operator+=(const Money &money)
{
  amt += money.Amount();
  return amt;
}
```

Implicit call to Conversion Constructor

This is because amountSaved is a Money

Call to += operator
Which creates a copy as in previous

Both copies immediately go out of scope
because they are not saved

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Printing account data" << std::endl;
balance = account->AmountSaved();
std::cout << "Account name: " << account->Owner() << std::endl;
std::cout << "Account id: " << account->Id() << std::endl;
std::cout << "Account balance: ";
PrintDollars(balance);
std::cout << std::endl;
```

Original value is replaced
So it goes out of scope

```
Printing account data
------Money Destructor (0x7ffca3ddb9b0)
Account name: Fred
Account id: 0
Account balance: $514.75
```

```
Money::~Money()
{
  std::cout << "------Money Destructor (" << this << ")\n";
}
```

# When Constructors Get Called

```
std::cout << std::endl;
std::cout << "Exiting program" << std::endl;
return 0;
```

```
Exiting program
------Money Destructor (0x7ffca3ddb9a0)
------Money Destructor (0x7ffca3ddb980)
------Money Destructor (0x7ffca3ddb970)
------Money Destructor (0x7ffca3ddb960)
------Money Destructor (0x7ffca3ddb950)
--Bank Destructor (0x7ffca3ddb9c0)
----Account Destructor (0x11cf0a0)
------Money Destructor (0x11cf0b0)
```

All variables were local to the main() function
When it finishes, a Destructor is run for each

```
Bank::~Bank()
{
  std::cout << "--Bank Destructor (" << this << ")\n";
  for (int i = 0; i < Bank::MAX_ACCOUNTS; i++)
    if (accounts[i] != NULL)
      delete accounts[i];
  delete accounts;
}
Account::~Account()
{
  std::cout << "----Account Destructor (" << this << ")\n";
}
```

Bank Destructor needs to remove Accounts
Each Account has an implicit Money Destructor