Passing Objects In Member Functions

Object Parameter

```
class Bank {
...
  int Withdraw(int id, const Money &money);
  int Deposit(int id, const Money &money);
...
};
```

- Why pass the object by reference?

Object Parameter

```
class Bank {
...
  int Withdraw(int id, const Money &money);
  int Deposit(int id, const Money &money);
...
};
```

- Why pass the object by reference?
  - Objects can potentially be very large and hold lots of different member variables
  - Passing by value invokes a constructor / destructor to copy each of the variables
  - This is especially taxing when dynamically allocating memory

Object Return

```
class Bank {
...
  Money AccountBalance(int id) const; //(1)
  Account* AccountData(int id) const; //(2)
  Account& AccountData(int id) const; //(3) not used
...
  Account **accounts;   //for 1 and 2
  Account accounts[10]; //for 3 (not used)
...
};
```

- What about the return type?
- Should it be a reference as well?

Object Return

```
class Bank {
...
  Money AccountBalance(int id) const; //(1)
  Account* AccountData(int id) const; //(2)
  Account& AccountData(int id) const; //(3) not used
...
  Account **accounts;   //for 1 and 2
  Account accounts[10]; //for 3
...
};
```

- What about the return type? It depends...
- In (1)
  - You are returning a local object
  - If you pass it by reference, it will cease to exist after it is returned
  - This is because it is local to the function and is created on the stack
  - However, it is still returned by value and a copy is created

Object Return

```
class Bank {
...
  Money AccountBalance(int id) const; //(1)
  Account* AccountData(int id) const; //(2)
  Account& AccountData(int id) const; //(3) not used
...
  Account **accounts;   //for 1 and 2
  Account accounts[10]; //for 3
...
};
```

- What about the return type? It depends...
- In (2)
  - You are returning a pointer
  - This is typically because the variable was dynamically generated so that it's scope would last beyond the function
  - This is more error prone, but avoids creating a copy

Object Return

```
class Bank {
...
  Money AccountBalance(int id) const; //(1)
  Account* AccountData(int id) const; //(2)
  Account& AccountData(int id) const; //(3) not used
...
  Account **accounts;   //for 1 and 2
  Account accounts[10]; //for 3
...
};
```

- What about the return type? It depends...
- In (3)
  - You are returning a reference
  - This should only be used if the object is a member variable
  - It is possible to pass local objects by reference...
    - But it is really error prone and messy

Implementation: Object Passing

```
int Bank::Deposit(int id, const Money &money)
{
  int i;
  for (i = 0; i < Bank::MAX_ACCOUNTS; i++)
    if (accounts[i] != NULL && accounts[i]->id == id)
      break;

  //could not find
  if (i >= Bank::MAX_ACCOUNTS)
    return -1;

  accounts[i]->amountSaved += money.Amount();
  return 0;
}
```

Used as if you passed by value

Just like with built-in types

Implementation: Return By Value

```
Money Bank::AccountBalance(int id) const
{
  int i;
  for (i = 0; i < Bank::MAX_ACCOUNTS; i++)
    if (accounts[i] != NULL && accounts[i]->id == id)
      break;

  //could not find
  if (i >= Bank::MAX_ACCOUNTS)
    return -1;

  return accounts[i]->amountSaved;
}
```

```
class Account
{
  ...
  Money amountSaved;
  ...
};
```

Returns a copy of the member variable

Don't worry about the -> operator yet
We'll cover it soon, just know that it's similar to the . operator

Implementation: Return By Pointer

```
Account* Bank::AccountData(int id) const
{
  int i;
  for (i = 0; i < Bank::MAX_ACCOUNTS; i++)
    if (accounts[i] != NULL && accounts[i]->id == id)
      break;

  //could not find
  if (i >= Bank::MAX_ACCOUNTS)
    return NULL;

  return accounts[i];
}
```

Returns a copy of the pointer:
`Account **accounts`

Implementation: Return By Reference

```
Account& Bank::AccountData(int id) const
{
  int i;
  for (i = 0; i < 10; i++)
    if (accounts[i].id == id)
      break;

  //could not find
  if (i >= Bank::MAX_ACCOUNTS)
    return NULL;

  return accounts[i];
}
```

Returns a reference to the member variable:
`Account accounts[10]`