CS 106X, Lecture 13
Unit Testing and Classes

reading:

Programming Abstractions in C++, Chapter 6

Plan For Today

- **Announcement:** Apply to Section Lead!
- Unit Testing
- Classes
  - What is a class?
  - Creating our own class

Announcement

- **Apply to Section Lead!**
- https://docs.google.com/presentation/d/1jLDpQalZYWjjGKkNa7C-OzadDMZxG0EAQjDFeiJE-K4

Plan For Today

- **Announcement:** Apply to Section Lead!
- Unit Testing
- Classes
  - What is a class?
  - Creating our own class

Unit Testing

Unit Testing

- **Unit Testing** is a method for testing small pieces or ”units” of source code in a larger piece of software.
- Each test is usually represented as a single function.
- **Key idea:** each test should examine one portion of functionality that is as narrow and isolated as possible.
- Each test has a way of indicating pass or failure.
- **Benefits:**
  - Limits your code to only what is necessary
  - Finds bugs early
  - Preserves functionality when code is changed

Unit Testing

- **expect()** ; displays error when condition inside this statement is false, but other tests continue to run.

Unit Testing

- **Example:** We are given a black-box function that takes a vector of ints and a number. It is supposed to return the largest sum we can make using elements in the vector without exceeding the number, and leave the passed-in vector unmodified.

Vector	Target Number	Return
{}	1	0
{10, 2, 5, 1}	9	8
{1, 2, 3, 4}	20	10

Unit Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!"

– Edsger Dijkstra

Plan For Today

- **Announcement:** Apply to Section Lead!
- Unit Testing
- Classes
  - What is a class?
  - Creating our own class

What if…

What if we could write a program like this:

```
BankAccount nickAccount("Nick", 25);
BankAccount zachAccount("Zach", 30);
nickAccount.deposit(10);
bool success = zachAccount.withdraw(10);
if (success) {
      cout << "Zach withdrew $10." << endl;
}
```

What if…

If we wanted to write that program with what we know so far:

```
string nickAccountName = "Nick";
double nickAccountBalance = 25;
string zachAccountName = "Zach";
double zachAccountBalance = 30;

nickAccountBalance += 10;
if (zachAccountBalance >= 10) {
      zachAccount.withdraw(10);
      cout << "Zach withdrew $10." << endl;
}
```

What If…

- *The first approach is much better:*
  - It encapsulates all of the logic about a bank account inside a single variable type
  - It hides away the complexity of bank accounts from the programmer using it

Classes

A class is a definition of your own custom variable type!

This Looks Familiar…

- A C++ *struct* is a way to define a new variable type that is a group of other variables.

```
struct Date {          // declaring a struct type
    int month;
    int day;           // members of each Date structure
};
...

Date today;            // construct structure instances
today.month = 10;
today.day = 23;

Date xmas {12, 25};    // shorter initializer syntax
```

Classes > Structs

- Structs default to public access for all members. Classes default to private members, which encourages the idea of *abstraction*: only exposing functionality and data that is important for the client to see.

Bank Account - Public

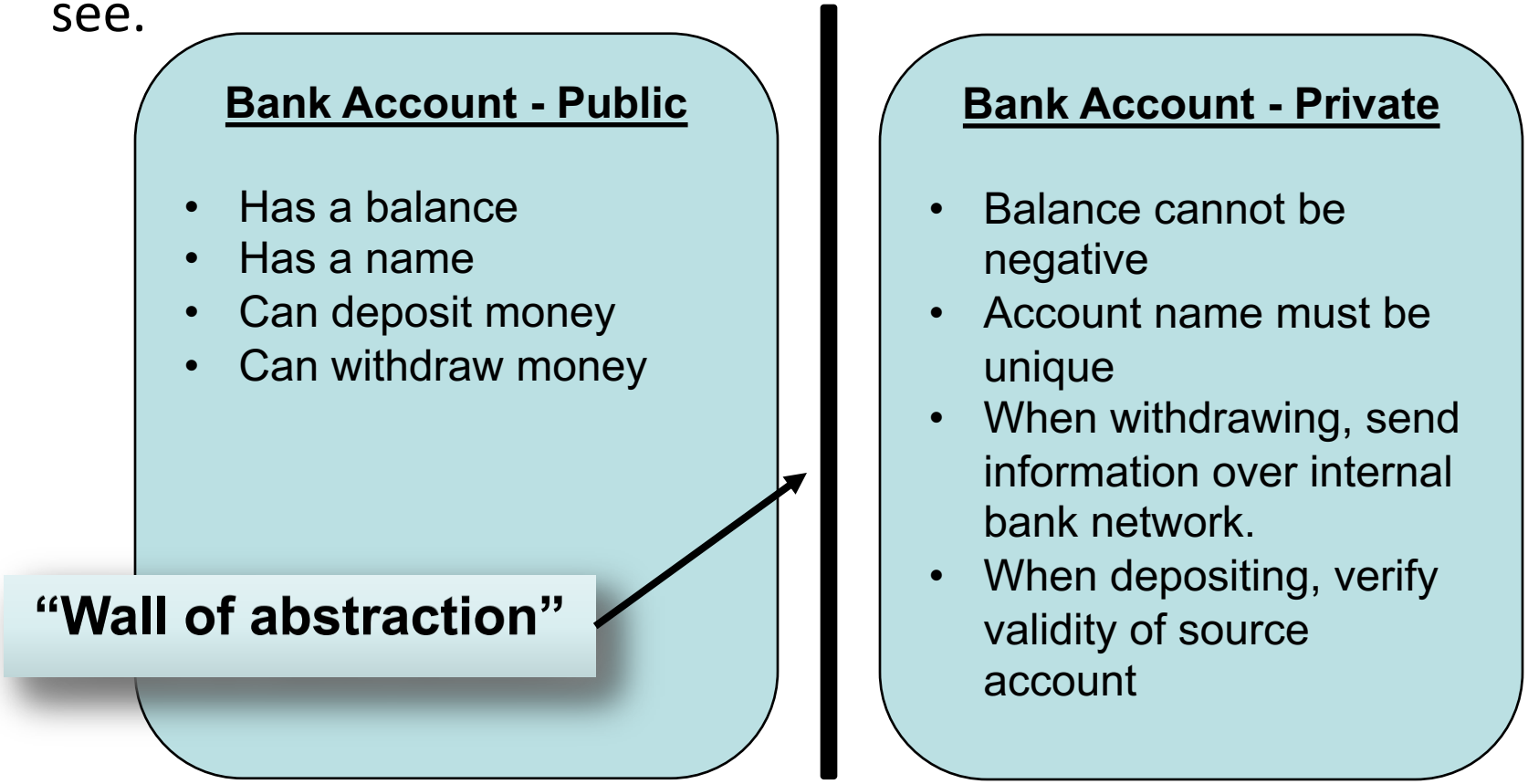
- Has a balance
- Has a name
- Can deposit money
- Can withdraw money

Bank Account - Private

- Balance cannot be negative
- Account name must be unique
- When withdrawing, send information over internal bank network.
- When depositing, verify validity of source account

Classes > Structs

- Structs default to public access for all members. Classes default to private members, which encourages the idea of *abstraction*: only exposing functionality and data that is important for the client to see.

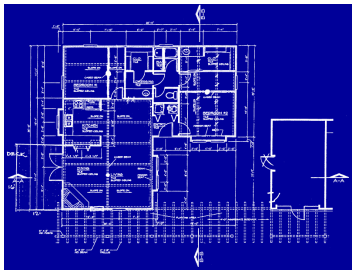


Classes Are Like Blueprints

iPod blueprint (class)

state:
- current song
- volume
- battery life

behavior:
- power on/off
- change station/song
- change volume
- choose random song



constructs

iPod (variable) #1

state:
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

behavior:
- power on/off
- change station/son
- change volume
- choose random song

iPod (variable) #2

state:
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

behavior:
- power on/off
- change station/son
- change volume
- choose random song

iPod (variable) #3

state:
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

The Classes Checklist

- ❑ **Specify instance variables.** What information is inside this new variable type?
- ❑ **Specify public methods**. What can this variable type do for others?
- ❑ **Specify constructor(s).** How do you create a new variable of this type?

Creating a New Class

- A class is defined across *two* files: a **.h** (header) file and a **.cpp** (source) file.

Interface

BankAccount.h

- Client can read through
- Shows methods and instance variables

Implementation

BankAccount.cpp

- Implementer writes
- Implements methods

Plan For Today

- **Announcement:** Apply to Section Lead!
- Unit Testing
- Classes
  - What is a class?
  - Creating our own class

Example: BankAccount

- Let's define a new variable type that represents a bank account.
- You should be able to create one by specifying the account name and the initial balance.
- You should be able to deposit and withdraw money, which should return whether or not that action was successful.
- You should also be able to obtain the account balance.

The Classes Checklist

- ❑ **Specify instance variables.** What information is inside this new variable type?
- ❑ **Specify public methods**. What can this variable type do for others?
- ❑ **Specify constructor(s).** How do you create a new variable of this type?

Example: BankAccount

- Our bank account variable type should store an instance variable for the name (string) and an instance variable for the balance (double).
- Instance variables should *always* be private. This way external files cannot modify them at will.

The Classes Checklist

✓**Specify instance variables.** What information is inside this new variable type?

❑**Specify public methods**. What can this variable type do for others?

❑**Specify constructor(s).** How do you create a new variable of this type?

Example: BankAccount

- We need a method for the following:
  - **bool deposit(amount)**: this should deposit the specified amount, and return whether it was successful. It is unsuccessful if the amount is negative.
  - **bool withdraw(amount)**: this should withdraw the specified amount, and return whether it was successful. It is unsuccessful if the account has insufficient funds.
  - **double getBalance()**: this should return the balance in the account.

Getters and Setters

Instance variables in a class should *always be private*. This is so only the object itself can modify them, and no-one else.

To allow the client to reference them, we define public methods in the class that **set** an instance variable's value and **get** (return) an instance variable's value. These are commonly known as **getters** and **setters**.

```
account.withdraw(25);
double balance = account.getBalance();
```

Getters and setters prevent instance variables from being tampered with.

The Classes Checklist

✓**Specify instance variables.** What information is inside this new variable type?

✓ **Specify public methods**. What can this variable type do for others?

❑**Specify constructor(s).** How do you create a new variable of this type?

Example: BankAccount

- To create a new bank account variable, the client must specify the name of the account and the initial amount.

.h Files

```
// classname.h
#pragma once

class ClassName {
    // class definition
};
```

#pragma once **basically says, "if you see this file more than once while compiling, ignore it after the first time" (so the compiler doesn't think you're trying to define things more than once)**

.h Files

```
// in ClassName.h
class ClassName {
public:
    ClassName(parameters);           // constructor
    returnType func1(parameters);    // member functions
    returnType func2(parameters);    // (behavior inside
    returnType func3(parameters);    //  each object)

private:
    type var1;     // member variables
    type var2;     // (data inside each object)
    type func4(); // (private function)
};
```

BankAccount.h

```
class BankAccount {
public:
    // Step 3: how to create a BankAccount
    BankAccount(string accountName, double startBalance);

    // Step 2: the things a BankAccount can do
    bool withdraw(double amount);
    bool deposit(double amount);
    double getBalance();
private:
    // Step 1: the data inside a BankAccount
    string name;
    double balance;
};
```

.cpp files

- In ***ClassName*.cpp**, we write bodies (definitions) for the member functions that were declared in the ***.h*** file:

```
// ClassName.cpp
#include "ClassName.h"

// member function
returnType ClassName::methodName(parameters) {
    statements;
}
```

  - Member functions/constructors can refer to the object's fields.

- *Exercise:* Write a `withdraw` member function to deduct money from a bank account's balance.

BankAccount.cpp

```
bool BankAccount::withdraw(double amount) {
   if (amount <= balance && amount >= 0) {
        balance -= amount;
     return true;
   }
   return false;
}

bool BankAccount::deposit(double amount) {
   if (amount >= 0) {
        balance += amount;
     return true;
   }
   return false;
}
```

BankAccount.cpp

```
double BankAccount::getBalance() {
    return balance;
}
```

Private data

```
private:
    type name;
```

- **encapsulation**: Hiding implementation details from client code.
- We can provide methods to get and/or set a data field's value:

```
// "read-only" access to the balance ("accessor")
double BankAccount::getBalance() const {
    return balance;
}

// Allows clients to change the field ("mutator")
void BankAccount::setName(string newName) {
    name = newName;
}
```

BankAccount.cpp

```
// Constructor
BankAccount::BankAccount(string accountName, double
                         startBalance) {
   name = accountName;
   balance = startBalance;
}

...
```

Recap

- **Announcement:** Apply to Section Lead!
- Unit Testing
- Classes
  - What is a class?
  - Creating our own class

Next time: *continuing with classes*