Playbook Notes: Introduction to Classes

DECLARING A CLASS:

You declare a **class** the same way you declare a structure, except instead of using the keyword struct, you will use the key word class

Ex/ (This a non-working example, we'll get to that in a minute)

```
class myClassType
{
    int value;
};
```

PRIVATE VS. PUBLIC

Assume the class above is declared, if we tried to access that value like we do with a struct, we would get an error.

```
int main(void)
{
    // A declaration of a class is called an object
    myClassType object;

    // This line will cause an error, hence, the red line
    object.value = 16;
```

The reason this causes an error, is class members are **private** by default and private member cannot be accessed outside the class. Structures are **public** by default and therefore require no special attention. To make a class member variable public, you must use the public keyword:

```
class myClassType
{
public:
    int value;
};
```

Now when you run the code in the main, you won't get an error:

But as a rule of thumb, all data in a class should ALWAYS be private. You can use the keyword private the same way you used public.

METHODS:

So how do you access the data if it's private? By using methods. A **method** is just a fancy word for a function that is declared inside a class.

So, lets declare the class again. This time using public, private, and a method:

```
class myClassType
{
public:
    // Method
    int GetValue(void);

private:
    int value;
};
```

Notice the following:

- The method is declared under the public keyword, so it can be accessed outside of the class, like the main.

- The method is just a prototype

- Our data, the integer value, is private

Fun Fact: All methods declared, have access to all the data in the class, so you will never have to pass it into the method. (This makes parameter passing MUCH easier!)

So now, let's declare our method:

DECLARING A METHOD (Part 1):

The following is a declaration of the GetValue method.

```
int myClassType::GetValue(void)
{
    return value;
}
```

Notice the following:

- In the header, you see myClassType:: before the function name. This tells C++ where the method is located. The :: is known as the **scope resolution operator**. It could also be used to call out a **namespace**.

- The method uses the integer variable, named value, from the class, but it is not declared inside the method or passed in as a parameter.

- The parameter list is void, but you can pass parameters just like you do in normal functions.

- Since this method does not change any data, it known as a getter method or an **Accessor method**.

You will call the method the same way you access a variable in a struct, with the **member operator** (.), or simply the dot.

```
int main(void)
{
    // A declaration of a class is called an object
    myClassType object;

    int localVariable;

    localVariable = object.GetValue();
```

Notice the following:

- The declaration of object, this is a declaration of our class, also known as an **object** Hence the name *Object Oriented Programming* or *OOP*.

- Also notice how the GetValue method is called and that it stores the value it returns in a local variable.

DECLARING A METHOD (Part 2):

Now let's re-create the class with another method:

```
class myClassType
{
public:
    // Methods
    int GetValue(void);

    // Our new method
    void SetValue(int someValue);

private:
    int value;
};
```

The method declaration:

```
void myClassType::SetValue(int someValue)
{
    // Assign our private member variable to the value of someValue
    value = someValue;
}
```

Notice the following:

- This method again, has access to the private data of the class.

- The private member variable value is assigned the value that is passed into the method.

- Because this method changes the data of the class, it is known as a setter method or a **mutator method**.

Next up, constructors and destructors

CONSTRUCTORS

Let's declare our class one more time, this time lets add a constructor:

```
class myClassType
{
public:

    // Declare a constructor
    myClassType(void);

    // Methods
    int GetValue(void);
    void SetValue(int someValue);

private:
    int value;
};
```

Notice the following about **constructors**:

- The constructor has the same name as the class, this is ALWAYS true for a constructor

- The constructor has no return type, this is intentional. A constructor CANNOT return a value

- The constructor is listed as a public method. The constructor must ALWAYS be public.

So how is a constructor called? The constructor is automatically called when you declare an object. It is also worth noting, that a constructor can be **overloaded**.

Example:

```
class myClassType
{
public:

    // Declare a constructor
    myClassType(void);

    // Declare an overloaded constructor
    myClassType(int someValue);

    // Methods
    int GetValue(void);
    void SetValue(int someValue);

private:
    int value;
};
```

Let's define both constructors:

```
myClassType::myClassType(int someValue)
{
    // Assign our private member variable to the value of someValue
    value = someValue;
}

myClassType::myClassType(void)
{
    // Assign our private member variable the default value of 0
    value = 0;
}
```

Notice the following:

- The constructor also has access to all private members

- The constructor declaration does *not* have a return type

- Constructors are often used to initialize data.

- When a class initiated, *only one* constructor will be called.

Now let's create two objects in the main, using each of the constructors:

```
int main(void)
{
    // A declaration of an object with the constructor
    // that has no parameters
    myClassType objectA;

    // A declaration of an object with the constructor
    // that has an integer parameter.
    myClassType objectB(16);
```

Notice the following:

- When I declare ObjectA, there are no parameters or ()

- Wen I declare ObjectB, I pass in an integer value

- I don't call the constructor, it is called automatically by the compiler

- If I was to call the GetValue method for ObjectA, it would return the value of 0

- If I was to call the GetValue method for ObjectB, it would return the value of 16

DESTRUCTORS

Constructors are created when an object is created, a destructor is called automatically when an object is destroyed. This typically happens when a program exits.

Let's add a destructor to our class:

```
class myClassType
{
public:

    // Constructors
    myClassType(void);
    myClassType(int someValue);

    // Destructor
    ~myClassType(void);

    // Methods
    int GetValue(void);
    void SetValue(int someValue);

private:
    int value;
};
```

Notice the following about **destructors**:

- The destructor has no return type, just like a constructor

- Like a constructor, the destructor is automatically called and cannot be called directly

- You see the tilde (~) in front of the class name, that is how you know it's a destructor

- Destructors cannot be over loaded, hence you cannot pass in parameters

If a constructor or destructor is not present in your class, C++ will create a default one for you.