Learning Objectives: Classes and Objects

- **Define the terms class, objects, instance, and instantiate**
- **Identify the difference between classes and objects**
- **Create a user-defined object**
- **Modify an object's attribute with dot notation**
- **Define a constructor**
- **Create a copy of an object**

Built-In Objects

The String Object and Others

You have already been using built-in C++ objects. Strings are an example of a C++ object.

```
string s = "I am a string";
cout << "s is a: " << typeid(s).name() << endl;
```

challenge

Try these variations:

Explore some of the functions associated with the string class.
* Add the line of code `cout << boolalpha << s.empty() << endl;`
* Add the line of code `cout << s.length() << endl;`
* Add the line of code `s.push_back('s');` and then `cout << s << endl;`

C++ says that the class or type of `s` is `NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE` (which is a string). Replace the exiting code with the one below and run the program again.

```
int arr[1];
cout << "arr is a: " << typeid(arr).name() << endl;
```

The resulting output says `arr is a: A1_i`. `A1` stands for one dimensional array and `i` stands for integer.

Vocabulary

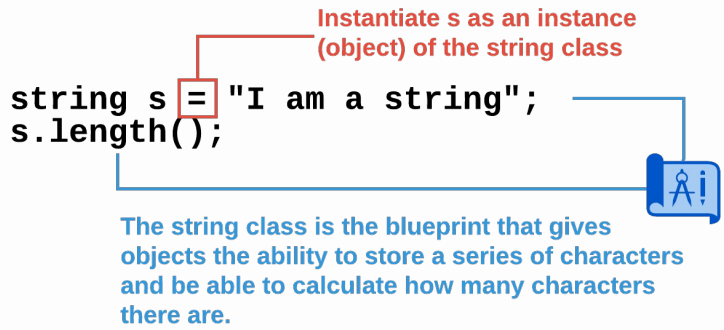
In the text above, the words “class” and “object” are used in an almost interchangeable manner. There are many similarities between classes and objects, but there is also an important difference. Working with objects has a lot of specialized vocabulary.

Classes - Classes are a collection of data and the actions that can modify the data. Programming is a very abstract task. Classes were created to give users a mental model of how to think about data in a more concrete way. Classes act as the blueprint. They tell C++ what data is collected and how it can be modified.

Objects - Objects are constructed according to the blueprint that is the class. In the code above, the variable `s` is a string object. It is not the class. The string class tells C++ that `s` has functions like `length`, `append`, and `replace`. When a programmer wants to use a class, they create an object.

Instance - Another way that programmers talk about objects is to say that an object is an instance of a particular class. For example, `s` is an instance of the `string` class.

Instantiation - Instantiation is the process where an object is created according to a blueprint of the class. The phrase “define a variable” means to create a variable. The variable is given a name and a value. Once it has been defined, you can use the variable. With objects, you use the phrase “instantiate an object”. That means to create an object, give it a name, store any data, and define the actions the object can perform.



.guides/img/objects/cpp_class_v_object.png

User-Defined Objects

Defining an Object

Assume you want to collect information about actors. Creating a class is a good way to keep this data organized. The `class` keyword is used to define a class. For now, do not add anything as the body of the class.

```
//add class definitions below this line

class Actor {

};

//add class definitions above this line
```

▼ **Naming classes**

The convention for naming classes in C++ is to use a capital letter. A lowercase letter will not cause an error message, but it is not considered to be "correct". If a class has a name with multiple words, all of the words are pushed together, and a capital letter is used for the first letter of each word. This is called camel case.

Classes are just the blueprint. To you use a class, you need to instantiate an object. Here is an object to represent Helen Mirren. Be sure to put this code in the `main` function.

```
//add code below this line

Actor helen;

//add code above this line
```

So you now have `helen`, which is now an instance of the `Actor` class.

Adding Attributes

The point of having a class is to collect information and define actions that can modify the data. The `Actor` class should contain things like the name of the actor, notable films, awards they have won, etc. These pieces of information related to a class are called attributes. Attributes are declared in the class itself. The example below adds the `first_name` and `last_name` attributes which are both strings. Notice that a keyword is also required `public`. `public` is considered to be an access specifier which determines how accessible the attributes are from outside the class. Adding this keyword just means that the attributes `first_name` and `last_name` are readily accessible. For now, we will be using `public` as the access specifier for our classes.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
};

//add class definitions above this line
```

You can change the value of an attribute with the assignment operator, `object_name.attribute = attribute_value`. Notice that you always use `object_name.attribute` to reference an attribute. This is called dot notation. Once an attribute has a value, you can treat it like any other variable. Add the following code to the `main` function. You are assigning values to the attributes `fist_name` and `last_name`, and then printing these values.

```
//add code below this line

Actor helen;
helen.first_name = "Helen";
helen.last_name = "Mirren";
cout << helen.first_name +  ' ' + helen.last_name << endl;

//add code above this line
```

challenge

Try these variations:

- Add the attribute `int total_films;` to the `class` and then assigned `helen.total_films = 80;` in `main`.
- Add the print statement `cout << helen.total_films << endl;` in `main`.
- Add the print statement `cout << helen << endl;`.

Note that many objects in C++ cannot be printed directly, thus `cout << helen << endl;` resulted in an error. When printing objects, be sure to reference their attributes.

The Constructor

Too Much Code

Imagine that the `Actor` class has more attributes than on the previous page.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
    string birthday;
    int total_films;
    int oscar_nominations;
    int oscar_wins;
};

//add class definitions above this line
```

Now create an object for Helen Mirren with values for each attribute. Adding each attribute individually requires several lines of code. This is especially true if you have more than one instance of the `Actor` class.

```
//add code below this line

Actor helen;
helen.first_name = "Helen";
helen.last_name = "Mirren";
helen.birthday = "July 26";
helen.total_films = 80;
helen.oscar_nominations = 4;
helen.oscar_wins = 1;
cout << helen.first_name + ' ' + helen.last_name << endl;

//add code above this line
```

The class `Actor` creates a class and its attributes. It does not assign value to any attributes; the user has to do this. A class is supposed to be a blueprint. It should lay out all of the attributes and their values for the user. Classes can do this when you use the constructor.

The Constructor

The constructor is a special function for a class. Its job is to assign value for attributes associated with the object. These attributes can also be called instance variables. In C++, the constructor is the class name, parentheses, and curly brackets. Inside of the constructor, give attributes their values.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
    string birthday;
    int total_films;
    int oscar_nominations;
    int oscar_wins;

  Actor() {
    first_name = "Helen";
    last_name = "Mirren";
    birthday = "July 26";
    total_films = 80;
    oscar_nominations = 4;
    oscar_wins = 1;
  }
};

//add class definitions above this line
```

Instantiating `helen` as an instance of the `Actor` class automatically calls the constructor. Since the instance variables (attributes) have values, you can remove those lines of code from the `main` function.

```
//add code below this line

Actor helen;
cout << helen.first_name + ' ' + helen.last_name << endl;

//add code above this line
```

challenge

Try these variations:

- Add this print statement to the `main` function:

```
cout << helen.first_name + ' ' + helen.last_name + "\'s
        birthday is " + helen.birthday + '.' << endl;
```

- Add this print statement to the `main` function:

```
cout << helen.first_name + ' ' + helen.last_name + " won " +
        helen.oscar_wins + " Oscar(s) out of " +
        helen.oscar_nominations + " nomination(s)." << endl;
```

- Change all of the `+` above to `<<`.

When you attempted to print `cout << helen.first_name + ' ' + helen.last_name + " won " + helen.oscar_wins + " Oscar out of " + helen.oscar_nominations + " nominations" << endl;`, you likely received a very long error message. This happened because the operator `+` only works with same-type objects or data. In the print statement above, you tried to combine strings with integers which C++ did not like and therefore complained. To solve this, simply change the `+` to `<<`.

The Constructor and Parameters

The Constructor and Parameters

Now imagine that you want to use the `Actor` class to instantiate an object for Helen Mirren and Tom Hanks. Create the `Actor` class just as before.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
    string birthday;
    int total_films;
    int oscar_nominations;
    int oscar_wins;

  Actor() {
    first_name = "Helen";
    last_name = "Mirren";
    birthday = "July 26";
    total_films = 80;
    oscar_nominations = 4;
    oscar_wins = 1;
  }
};

//add class definitions above this line
```

Now instantiate two `Actor` objects, one for Helen Mirren and the other for Tom Hanks. Print the `fist_name` and `last_name` attributes for each object.

```
//add code below this line

Actor helen;
Actor tom;
cout << helen.first_name << ' ' << helen.last_name << endl;
cout << tom.first_name << ' ' << tom.last_name << endl;

//add code above this line
```

The constructor `Actor` class only creates an object with information about Helen Mirren. You can make the `Actor` class more flexible by passing it an argument for each of attributes in the constructor. Parameters for the constructor function work just as they do for user-defined functions, be sure to indicate the data type for each parameter.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
    string birthday;
    int total_films;
    int oscar_nominations;
    int oscar_wins;

  Actor(string fn, string ln, string bd, int tf, int on, int ow)
        {
    first_name = fn;
    last_name = ln;
    birthday = bd;
    total_films = tf;
    oscar_nominations = on;
    oscar_wins = ow;
  }
};

//add class definitions above this line
```

When you instantiate the two `Actor` objects, you can pass the constructor the relevant information for both Helen Mirren and Tom Hanks. The code should now print the correct first and last names.

```
//add code below this line

Actor helen("Helen", "Mirren", "July 26", 80, 4, 1);
Actor tom("Tom", "Hanks", "July 9", 76, 5, 2);
cout << helen.first_name << ' ' << helen.last_name << endl;
cout << tom.first_name << ' ' << tom.last_name << endl;

//add code above this line
```

challenge

Try these variations:

- Create an instance of the `Actor` class for Denzel Washington (December 28, 47 films, 8 nominations, 2 wins).
- Print the `birthday` and `total_films` attributes for the newly created object.

▼ **Code**

Your code for the object representing Denzel Washington should look something like this:

```
      //add code below this line

      Actor denzel("Denzel", "Washington", "December 28",
        47, 8, 2);
      cout << denzel.birthday << endl;
      cout << denzel.total_films << endl;

      //add code above this line
```

Default Values

We can assume that the average actor probably has not been nominated or won an Oscar. So instead of making these attributes parameters for the constructor, we can give them the default value of 0. These attributes can always be updated later on.

```
//add class definitions below this line

class Actor {
  public:
    string first_name;
    string last_name;
    string birthday;
    int total_films;
    int oscar_nominations;
    int oscar_wins;

  Actor(string fn, string ln, string bd, int tf) {
    first_name = fn;
    last_name = ln;
    birthday = bd;
    total_films = tf;
    oscar_nominations = 0;
    oscar_wins = 0;
  }
};

//add class definitions above this line
```

You can update the attributes once the object has been instantiated if need be.

```
//add code below this line

Actor helen("Helen", "Mirren", "July 26", 80);
cout << helen.oscar_nominations << endl;
cout << helen.oscar_wins << endl;

helen.oscar_nominations = 4;
helen.oscar_wins = 1;

cout << helen.oscar_nominations << endl;
cout << helen.oscar_wins << endl;

//add code above this line
```

Copying Objects

Copying an Object

In C++, you can initialize a new object to an existing one to create a clone. What do you thin the code below does?

```
//add class definitions below this line

class ComicBookCharacter {
  public:
    string name;
    int age;
    string type;
};

//add class definitions above this line

int main() {

  //add code below this line

  ComicBookCharacter a;
  a.name = "Calvin";
  a.age = 6;
  a.type = "human";

  ComicBookCharacter b = a;
  a.name = "Hobbes";

  cout << "Object a name: " << a.name << endl;
  cout << "Object a age: " << a.age << endl;
  cout << "Object a type: " << a.type << endl;
  cout << "Object b name: " << b.name << endl;
  cout << "Object b age: " << b.age << endl;
  cout << "Object b type: " << b.type << endl;

  //add code above this line
```

You'll notice that the initializing one object to another created an exact copy of the original object. Each object is still treated as separate objects though so you can still freely change the attribute of one object without

affecting another.

challenge

Try these variations:

- Add `b.name = "Snoopy";` before the print statements.
- Add `b.type = "dog";` before the print statements.