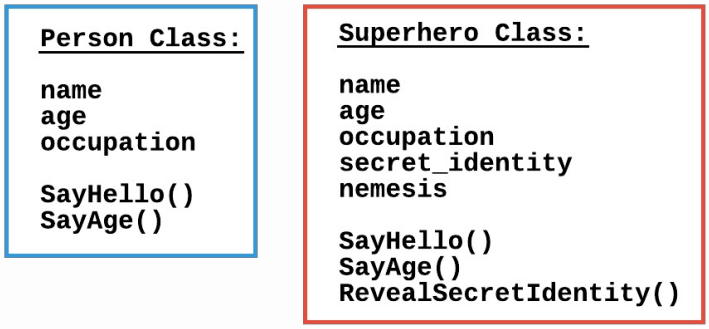
Learning Objectives: Extending & Overriding

- **Define the terms extending and overriding**
- **Extend the derived class with a new function**
- **Override functions from the base class with new functionality**

Extending a Class

Extending the Derived Class

The idea of inheritance is to borrow from a base class and then add on functionality. Up until now, we have talked about borrowing from a base class but have not gone into detail about adding additional functionality to the derived class. The process of adding functionality to a derived class is known as either extending or overriding. Extending a class means that new attributes and functions are given to the derived class. Let's continue working with our `Person` and `Superhero` classes.



.guides/img/inheritance/CppNoInheritance

The code below will first associate the derived class constructor with the base class constructor. However, we want to add an additional attribute as a parameter to the derived class constructor. Doing so will extend the derived class because objects created will have 4 parameter attributes instead of 3. Additional getter and setter functions are also added to extend the derived class even further.

```
//add class definitions below this line

class Superhero : public Person {
  public:
    Superhero(string n, int a, string o, string s) : Person(n,
        a, o) {
      secret_identity = s;
    }

    string GetSecretIdentity() {
      return secret_identity;
    }

    void SetSecretIdentity(string new_secret_identity) {
      secret_identity = new_secret_identity;
    }

  private:
    string secret_identity;
};

//add class definitions above this line
```

In `main`, instantiate a `Superhero` object and print out each of the attributes. You should see the three attributes from the `Person` class as well as the new attribute `secret_identity`.

```
  //add code below this line

  Superhero hero("Spider-Man", 16, "student", "Peter Parker");
  cout << hero.GetName() << endl;
  cout << hero.GetAge() << endl;
  cout << hero.GetOccupation() << endl;
  cout << hero.GetSecretIdentity() << endl;

  //add code above this line
```

▼ Inheritance Is a One-Way Street

Inheritance shares attributes and functions from the base class to the derived class. When a derived class is extended, it cannot share the new additions with its base class. For example, in the code above, the `Superhero` class has access to the attributes `name`, `age`, and `occupation`, but `Person` does not have access to `secret_identity`.

challenge

Try this variation:

- Rewrite the `Superhero` class so that it extends the `Person` class by adding the string attribute `nemesis`. The constructor will also include `nemesis` as the last parameter with `Doc Octopus` as its argument. Remember to include the relevant getter and setter functions too!

  ▼ **Solution**

```
//add class definitions below this line

class Superhero : public Person {
  public:
    Superhero(string n, int a, string o, string s, string
        ne) : Person(n, a, o) {
      secret_identity = s;
      nemesis = ne;
    }

    string GetSecretIdentity() {
      return secret_identity;
    }

    void SetSecretIdentity(string new_secret_identity) {
      secret_identity = new_secret_identity;
    }

    string GetNemesis() {
      return nemesis;
    }

    void SetNemesis(string new_nemesis) {
      nemesis = new_nemesis;
    }

  private:
    string secret_identity;
    string nemesis;
};

//add class definitions above this line

int main() {
```

```
  //add code below this line

  Superhero hero("Spider-Man", 16, "student", "Peter
        Parker", "Doc Octopus");
  cout << hero.GetName() << endl;
  cout << hero.GetAge() << endl;
  cout << hero.GetOccupation() << endl;
  cout << hero.GetSecretIdentity() << endl;
  cout << hero.GetNemesis() << endl;

  //add code above this line

  return 0;

}
```

Extending a Class by Adding Unique Functions

Another way to extend a class is to create new functions that are unique to the derived class (besides getter and setter functions). Currently, the function `SayHello` will print the superhero's name, but it will not print their secret identity. Create the function `RevealSecretIdentity` to print a greeting that reveals `SecretIdentity`.

```
//add class definitions below this line

class Superhero : public Person {
  public:
    Superhero(string n, int a, string o, string s, string ne) :
        Person(n, a, o) {
      secret_identity = s;
      nemesis = ne;
    }

    string GetSecretIdentity() {
      return secret_identity;
    }

    void SetSecretIdentity(string new_secret_identity) {
      secret_identity = new_secret_identity;
    }

    string GetNemesis() {
      return nemesis;
    }

    void SetNemesis(string new_nemesis) {
      nemesis = new_nemesis;
    }

    void RevealSecretIdentity() {
      cout << "My real name is " << secret_identity << '.' <<
        endl;
    }

  private:
    string secret_identity;
    string nemesis;
};

//add class definitions above this line
```

Now test out the newly added function.

```
  //add code below this line

  Superhero hero("Spider-Man", 16, "student", "Peter Parker",
        "Doc Octopus");
  hero.RevealSecretIdentity();

  //add code above this line
```

challenge

Try this variation:

- Create the function `SayNemesis` that prints the string `My nemesis is Doc Octopus.`, then call it on `hero` in `main`.
  ▼ **Solution**

```
//add class definitions below this line

class Superhero : public Person {
  public:
    Superhero(string n, int a, string o, string s, string
        ne) : Person(n, a, o) {
      secret_identity = s;
      nemesis = ne;
    }

    string GetSecretIdentity() {
      return secret_identity;
    }

    void SetSecretIdentity(string new_secret_identity) {
      secret_identity = new_secret_identity;
    }

    string GetNemesis() {
      return nemesis;
    }

    void SetNemesis(string new_nemesis) {
      nemesis = new_nemesis;
    }

    void RevealSecretIdentity() {
      cout << "My real name is " << secret_identity << '.'
        << endl;
    }

    void SayNemesis() {
      cout << "My nemesis is " << nemesis << '.' << endl;
    }

  private:
    string secret_identity;
    string nemesis;
```

```
};

//add class definitions above this line

int main() {

  //add code below this line

  Superhero hero("Spider-Man", 16, "student", "Peter
        Parker", "Doc Octopus");
  hero.SayNemesis();

  //add code above this line

  return 0;

}
```

Function Overriding

Overriding a Function

Extending a class means adding new attributes or functions to the derived class. Another way to add new functionality to a derived class is through function overriding. Overriding a function means to inherit a function from the base class, keep its name, but change the contents of the function.

Extend the `Superhero` class by overriding the `SayHello` function. Remember, the `name` attribute is part of the base class and it is private, so you need to use the `GetName` function to access this attribute.

```
  //add class definitions below this line

  void SayHello() {
    cout << "My name is " << GetName() << ", and criminals
      fear me." << endl;
  }

  //add class definitions above this line
```

Instantiate a `Superhero` object and call the `SayHello` function on it.

```
//add code below this line

Superhero hero("Storm", 30, "Queen of Wakanda", "Ororo
      Munroe", "Shadow King");
hero.SayHello();

//add code above this line
```

▼ **Differentiating Overriding and Extending**

The difference between extending and overriding can be slight. Both approaches are used to make a derived class unique from the base class. However, overriding deals with **changing** a pre-existing function from the base class, while extending deals with **adding** new functions and attributes.

challenge

Try this variation:

- Add and override the `SayAge` function in the `Superhero` class so that it prints the string, `Age is just a number.`, then call it on `hero` in the `main` function.

  ▼ **Solution**

```
void SayAge() {
  cout << "Age is just a number." << endl;
}
```

```
Superhero hero("Storm", 30, "Queen of Wakanda", "Ororo
        Munroe", "Shadow King");
hero.SayAge();
```

What Happens to the Overridden Function?

If you can override a function from the base class, what happens to its original function? C++ defaults to the instance or object type. So `hero.SayHello()` will always use the function from the derived `Superhero` class. But that does not mean you cannot call `SayHello` from the base `Person` class. To call the original base class function, you can use `hero.Person::SayAge()` where `hero` represents the derived class object, `Person` represents the base class, and `SayAge` represents the base class function. The `::` is called the **scope resolution operator** and it is used to direct C++ to look for the function `SayAge` inside the `Person` class. Make sure you have the following class definitions in your code.

```
  //add class definitions below this line

  void SayHello() {
    cout << "My name is " << GetName() << ", and criminals
      fear me." << endl;
  }

  void SayAge() {
    cout << "Age is just a number." << endl;
  }

  //add class definitions above this line
```

Then run the following commands in `main` to see the result.

```
//add code below this line

Superhero hero("Storm", 30, "Queen of Wakanda", "Ororo
      Munroe", "Shadow King");
hero.SayHello();
hero.Person::SayHello();
hero.SayAge();
hero.Person::SayAge();

//add code above this line
```

Note how in the code above, using the scope resolution operator causes the base class function to be called while not using it causes the derived class function to be called.

challenge

Try this variation:

- Modify the following code in the `Person` class:

```
void SayHello() {
  cout << "Hello, my name is " << name << '.' << endl;
}

void SayAge() {
  cout << "I am " << age << " years old." << endl;
}
```

to

```
virtual void SayHello() final {
  cout << "Hello, my name is " << name << '.' << endl;
}

virtual void SayAge() final {
  cout << "I am " << age << " years old." << endl;
}
```

▼ **Why is there an error?**

To prevent a derived class from overriding a base class function, you can use the key terms `virtual` and `final`. `virtual` goes in front of the function declaration name while `final` goes behind. Adding these key terms keeps you from overriding the functions in the `Superhero` class. That is why you see the error. You'll learn more about these key terms in a future module.