Child Classes

by Sophia

☰ WHAT'S COVERED

In this lesson, you will explore how subclasses are used and how they can extend functionality. Specifically, this lesson covers:

1. Customizing Subclasses

1. Customizing Subclasses

So far, you have learned that a subclass accepts all the different parameters of its base class. In a previous example, you created and defined a base class called `Member`. You then created two subclasses called `Admin` and `User`. The subclasses inherited the methods from the base class, and these, in turn, provided access to the data in the attributes. However, the `Admin` and `User` were just subclasses without any unique characteristics. If you recall, you declared the subclasses with parameterized constructors that just called the parent class's parameterized constructor using **super()**.

Let's look at the prior example again, starting with the `Member` class (in a file named Member.java).

```
import java.time.LocalDate;

public class Member {
  private String firstName;
  private String lastName;
  private int expiryDays = 365;
  private LocalDate expiryDate;

  public Member(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    expiryDate = LocalDate.now().plusDays(expiryDays);
  }
```

```java
  public String getFirstName() {
    return firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public LocalDate getExpiryDate() {
    return expiryDate;
  }
}
```

⇨ EXAMPLE Here is the code for the `Admin` subclass (in Admin.java):

```java
public class Admin extends Member{
  public Admin(String firstName, String lastName) {
    super(firstName, lastName);
  }
}
```

⇨ EXAMPLE And here is the almost identical code for the `User` class (User.java):

```java
public class User extends Member {
  public User(String firstName, String lastName) {
    super(firstName, lastName);
  }
}
```

⇨ EXAMPLE Lastly, here is the driver class that is used to test these classes (MemberInheritance.java):

```java
public class MemberInheritance {
  public static void main(String[] args) {
    Member testMember = new Member("Sophia", "Java");
    System.out.println("First Name: " + testMember.getFirstName());
    System.out.println("Last Name: " + testMember.getLastName());
    System.out.println("Exp. Date: " + testMember.getExpiryDate());

    Admin testAdmin = new Admin("root", "admin");
    System.out.println("First Name: " + testAdmin.getFirstName());
    System.out.println("Last Name: " + testAdmin.getLastName());
    System.out.println("Exp. Date: " + testAdmin.getExpiryDate());
```

```
    User testUser = new User("Artie", "Smith");
    System.out.println("First Name: " + testUser.getFirstName());
    System.out.println("Last Name: " + testUser.getLastName());
    System.out.println("Exp. Date: " + testUser.getExpiryDate());
  }
}
```

Running the driver class produced output like this:

```
First Name: Sophia
Last Name: Java
Exp. Date: 2025-05-07
First Name: root
Last Name: admin
Exp. Date: 2025-05-07
First Name: Artie
Last Name: Smith
Exp. Date: 2025-05-07
```

There is nothing specific defined in either subclass other than the constructor (and its call to the `super()`). To truly make the subclasses useful, we need them to have some differences.

One of the things we can do is to make an attribute that has a default value different from what's in the base class. For example, in the `Member` class, we set the `expiryDays` attribute to 365. However, if we want a rule in place that we don't want the `Admin` accounts to expire until 100 years from now, we could change the `expiryDays` value to be 365 * 100. We do this just by declaring the attribute with the desired default value in the `Admin` subclass. Since the `getExpiryDate()` method inherited from the base class will return the expiration date for a `Member`, we also need to override the `getExpiryDate()` method in the `Admin` subclass.

⬚ TRY IT

Directions: Enter this code into the IDE and make sure you get the same output before moving on:

```
import java.time.LocalDate;
// Subclass of Member for administrators
public class Admin extends Member{
  private int expiryDays = 100 * 365;
  private LocalDate expiryDate;

  public Admin(String firstName, String lastName) {
    super(firstName, lastName);
    expiryDate = LocalDate.now().plusDays(expiryDays);
  }
```

```
  @Override
  public LocalDate getExpiryDate() {
    return expiryDate;
  }
}
```

The private `expiryDays` attribute in the base class is not inherited, so you can declare an `expiryDays` attribute in the subclass, but to get the public `getExpiryDays()` method to return the correct value for the subclass (rather than the base class), you would need to override the inherited `getExpiryDate()` method.

Running the MemberInheritance.java program should now produce results like these (the specific dates will vary):

```
First Name: Sophia
Last Name: Java
Exp. Date: 2025-05-07
First Name: root
Last Name: admin
Exp. Date: 2124-04-13
First Name: Artie
Last Name: Smith
Exp. Date: 2025-05-07
```

🗹 TRY IT

Directions: Try modifying the Admin subclass as shown above with the following updates:

Sometimes a subclass has attributes that the base class does not. In that case, you may want to pass an argument to the subclass's constructor that doesn't exist in the base class. Doing so is a little more complicated; however, it is a common technique, so you should be aware of the steps to do that.

The `Admin` subclass already has its own constructor that calls the base class's constructor using `super()` and then initializes the attributes that are specific to the `Admin` class. In this case, you would need to add a `String` attribute named secret with a line like this:

⇗ EXAMPLE

```
   private String secret;
```

This attribute does not have a default value, so the constructor for the `Admin` class will need to be modified to add a parameter to provide this value when an `Admin` object is created. The constructor's signature line now will look like this:

```
public Admin(String firstName, String lastName, String secret) {
```

See, it looks identical to the base class's constructor except for the last parameter named `secret`.

At this point, it's not important what the value of secret will be used for, but if the value is to be used for something, we will need to provide an accessor method:

```
public String getSecret() {
 return secret;  }
```

📝 TRY IT

Directions: Revise the Admin class as shown above:

The revised version of the `Admin` class should now look like this:


```java
import java.time.LocalDate;

// Subclass of Member for administrators
public class Admin extends Member{
  private int expiryDays = 100 * 365;
  private LocalDate expiryDate;
  private String secret;

  public Admin(String firstName, String lastName, String secret) {
    super(firstName, lastName);
    expiryDate = LocalDate.now().plusDays(expiryDays);
    this.secret = secret;
  }

  @Override
  public LocalDate getExpiryDate() {
    return expiryDate;
  }

  public String getSecret() {
    return secret;
```

```
    }
}
```

It is important to remember that while subclasses get access to the public methods in the base class (and thus get access to the attributes in the base class) for free, this doesn't mean that the subclass is limited to these. Subclasses can build on the functionality provided by the base class and customize it.

Let's test this altogether now with an updated `Admin` subclass and updated `Admin` instance using a new third argument for the secret parameter. We are leaving the `User` subclass as it was. Note: we placed a string of lines to separate each subclass for easier viewing.

[✎] TRY IT

Directions: To test these classes, use the following code for the AdminUserTest class (saved in a file named AdminUserTest.java):

```
class AdminUserTest {
  public static void main(String[] args) {
    Admin testAdmin = new Admin("root", "admin", "ABRACADABRA");
    System.out.println("First Name: " + testAdmin.getFirstName());
    System.out.println("Last Name: " + testAdmin.getLastName());
    System.out.println("Secret code: " + testAdmin.getSecret());
    System.out.println("Expiry Date: " + testAdmin.getExpiryDate());

    System.out.println("--------------------");

    User testUser = new User("Artie", "Smith");
    System.out.println("First Name: " + testUser.getFirstName());
    System.out.println("Last Name: " + testUser.getLastName());
    System.out.println("Expiry Date: " + testUser.getExpiryDate());
  }
}
```

[✎] TRY IT

Directions: Enter the updated code to test the application. We are only creating instances of the subclasses. Run the AdminUserTest.java program:

The output for this should look something like the following:

```
First Name: root
Last Name: admin
Secret code: ABRACADABRA
```

```
Expiry Date: 2124-04-13
-------------------
First Name: Artie
Last Name: Smith
Expiry Date: 2025-05-07
```

In this example, we have created an instance of the `Admin` subclass with arguments of "root" for firstname, "admin" as lastname, and "ABRACADABRA" as the secret code value.

In the output, we can see that all the attributes are present, including the updated attribute `expiryDate` from the `Admin` subclass. You can also see that the `User` subclass hasn't been affected by any changes that were made to the `Admin` subclass.

⬚ TRY IT

Directions: Test this by trying to call getSecret() from the User subclass using code like the following:

⤳ EXAMPLE

```
User testUser = new User("Artie", "Smith");
System.out.println("First Name: " + testUser.getFirstName());
System.out.println("Last Name: " + testUser.getLastName());
System.out.println("Secret code: " + testUser.getSecret());
System.out.println("Expiry Date: " + testUser.getExpiryDate());
```

Note that this code will not compile or run as demonstrated below:

```
AdminUserTest.java:14: error: cannot find symbol
    System.out.println("Secret code: " + testUser.getSecret());
                                                  ^
  symbol:   method getSecret()
  location: variable testUser of type User
1 error
error: compilation failed
```

❓ REFLECT

It is important to keep in mind that subclasses "know" about the parent class, but the parent (or base) class doesn't "know" about the functionality that the subclasses add. While classes inheriting from a common base class have common functionality, they may customize or build on the base class in different ways that do not affect each other.

This is because the `getSecret()` method only belongs to the Admin subclass and not the User subclass. What we define in a subclass is not inherited by or added to the other subclasses of the same base class.

There are also instances where we may have the same method name in the base class and in a subclass. We saw this when we overrode the inherited `getExpiryDate()` method in the Admin subclass. When a subclass overrides an inherited method, Java will use the most specific one that's tied to the subclass. It will use the more generic method if nothing in that subclass overrides the method in the base class.

For example, we'll add a method called `getStatus()` to the base class and each subclass. The `getStatus()` method returns a concatenated string with first name, last name, and the class.

Here is the method as it needs to be implemented in the `Member` class:

⇗ EXAMPLE

```
public String getStatus() {
  return firstName + " " + lastName + " is a Member.";
}
```

Since the `firstName` and `lastName` are private attributes in the Member class, they can be referenced by name in the method. In the subclasses, though, the methods that override the `getStatus()` method will need to use the `getFirstName()` and `getLastName()` methods inherited from the Member class.

Here is the version of the method for the `Admin` and `User` classes:

⇗ EXAMPLE

```
public String getStatus() { return getFirstName() + " " + getLastName() + " is an Admin."; }
```

🖉 TRY IT

Directions: Modify the code in the MemberInheritance.java file so that it reads as follows:

```
public class MemberInheritance {
  public static void main(String[] args) {
    Member testMember = new Member("Sophia", "Java");
    System.out.println("First Name: " + testMember.getFirstName());
    System.out.println("Last Name: " + testMember.getLastName());
    System.out.println("Exp. Date: " + testMember.getExpiryDate());
    System.out.println(testMember.getStatus());
    System.out.println("--------------------");

    Admin testAdmin = new Admin("root", "admin", "ABRACADABRA");
    System.out.println("First Name: " + testAdmin.getFirstName());
    System.out.println("Last Name: " + testAdmin.getLastName());
```

```
        System.out.println("Exp. Date: " + testAdmin.getExpiryDate());
        System.out.println(testAdmin.getStatus());
        System.out.println("-------------------");

        User testUser = new User("Artie", "Smith");
        System.out.println("First Name: " + testUser.getFirstName());
        System.out.println("Last Name: " + testUser.getLastName());
        System.out.println("Exp. Date: " + testUser.getExpiryDate());
        System.out.println(testUser.getStatus());
    }

}
```

[✎ TRY IT]

Directions: After entering the new code for the `MemberInheritance` driver class, run the program:

The results should look like this (though the dates will vary):

```
First Name: Sophia
Last Name: Java
Exp. Date: 2025-05-07
Sophia Java is a Member.
-------------------
First Name: root
Last Name: admin
Exp. Date: 2124-04-13
root admin is a Member.
-------------------
First Name: Artie
Last Name: Smith
Exp. Date: 2025-05-07
Artie Smith is a Member.
```

[✎ TRY IT]

Directions: Let's comment out the `getStatus()` method from the `User` class in the User.java file so that it reads like this:

```
public class User extends Member {
  public User(String firstName, String lastName) {
    super(firstName, lastName);
  }

  /* @Override
```

```
  public String getStatus() {
    // Need to use inherited methods to get name
    return getFirstName() + " " + getLastName() + " is a User.";
  } */
}
```

Directions: Run the program (MemberInheritance):

Notice that Artie Smith is showing the status of `Member` instead of `User`.

```
First Name: Sophia
Last Name: Java
Exp. Date: 2025-05-07
Sophia Java is a Member.
-------------------
First Name: root
Last Name: admin
Exp. Date: 2124-04-13
root admin is a Member.
-------------------
First Name: Artie
Last Name: Smith
Exp. Date: 2025-05-07
Artie Smith is a Member.
```

That's because the getStatus() method is no longer overridden in the User class. As such, the Java compiler has to look in the Member class (base class) and use the one found there.

☁   **BRAINSTORM**

Directions: Try removing the method from the "User" class (or comment it out) and see if you get the same output (though the dates will vary).

🗎   **TERM TO KNOW**

super()
The super() method allows the constructor in a subclass to call the constructor for the base class.

🗒   **SUMMARY**

In this lesson, you learned that subclasses do not only have to use the methods that they inherit from the base class. **Subclasses can be customized** or extended from the base class. To do this, subclasses

can have their own constructors. With the use of the super() method, a subclass can call the base class's constructor. You were able to build out subclasses with extra attributes and methods. We also learned that if a method is the same in the base class and subclasses, Java will use the most specific one that's tied to the subclass. It will use the base class method if nothing in that subclass has that method name.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source **cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf**

It has also been adapted from "Python for Everybody" By Dr. Charles R. Severance. Source **py4e.com/html3/**

📄	TERMS TO KNOW

super()

The super() method allows the constructor in a subclass to call the constructor for the base class.