Introduction to Inheritance

by Sophia

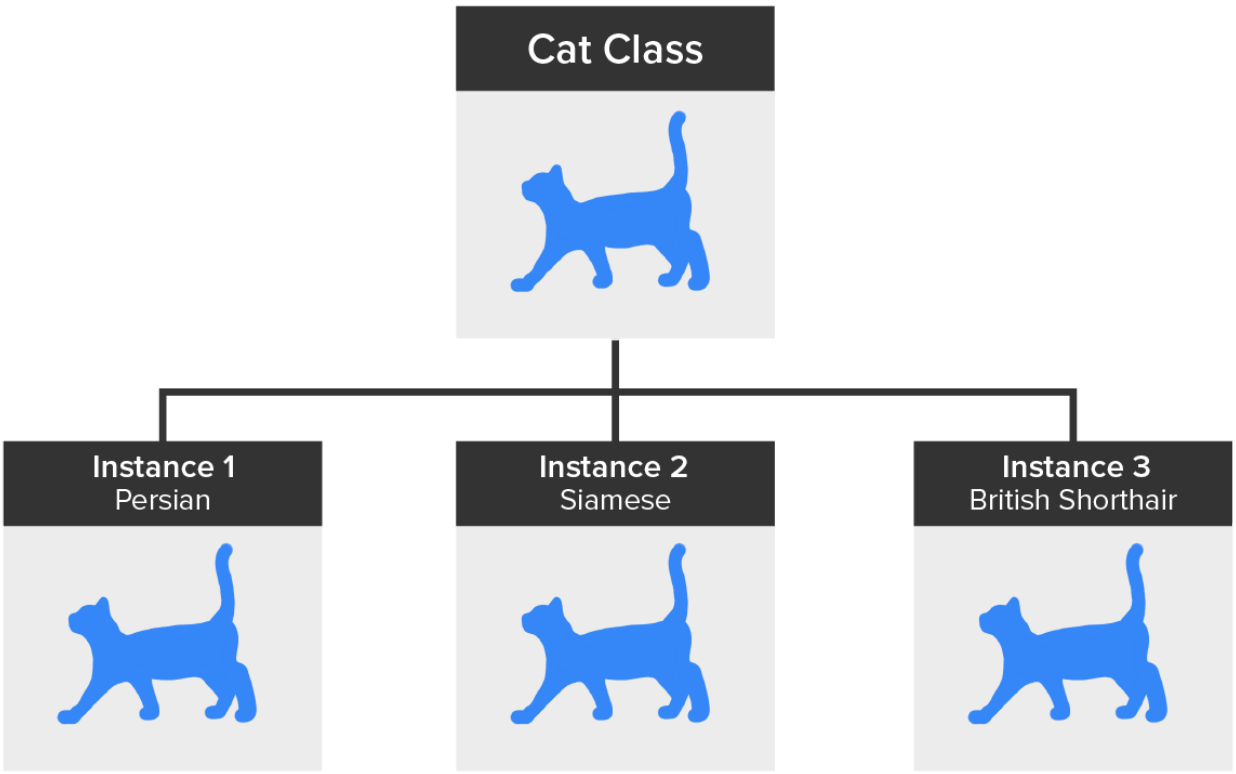
WHAT'S COVERED

In this lesson, you will learn about inheritance and how properties and methods are inherited from a parent class. Specifically, this lesson covers:

1. Real-World Example

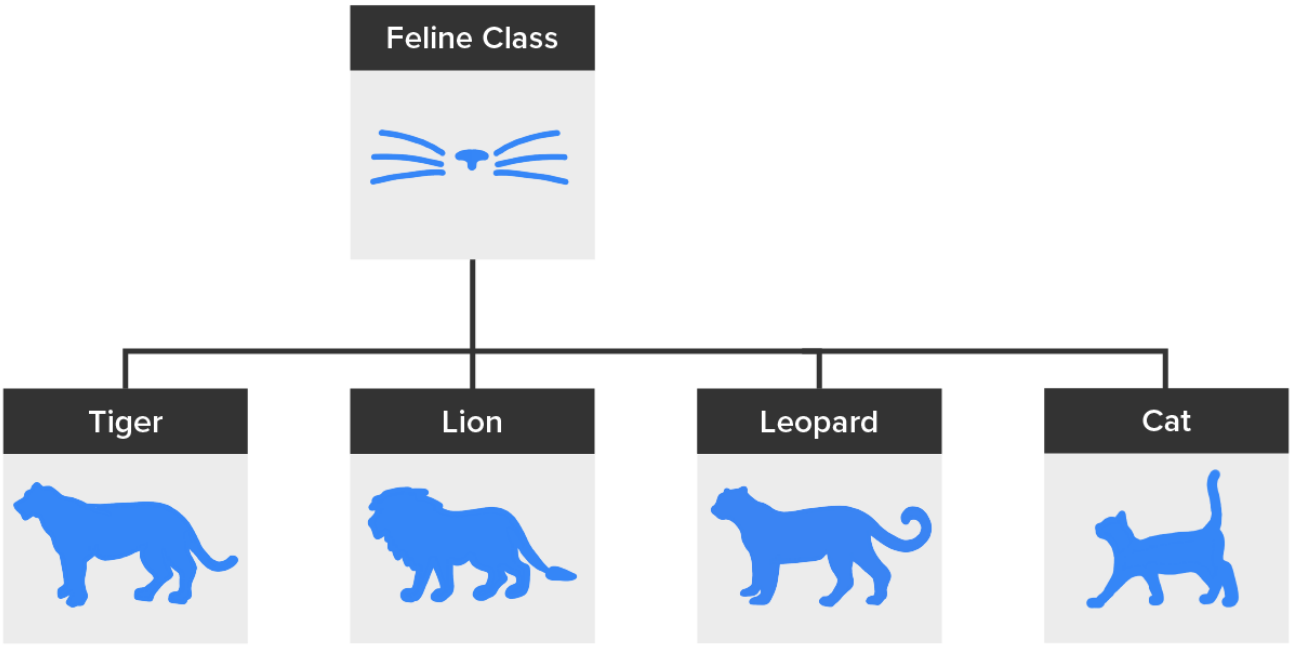
Inheritance and subclasses are often topics of conversation when discussing object-oriented programming. While these terms may sound complex, these are concepts about things that you actually see in the real world.

If you want to consider cats as a Java class, you could put together all of the possible cats into a class of animals that you would call `Cats`. Even though each cat may be unique, it is still a type of cat and would be a member of your `Cat` class.



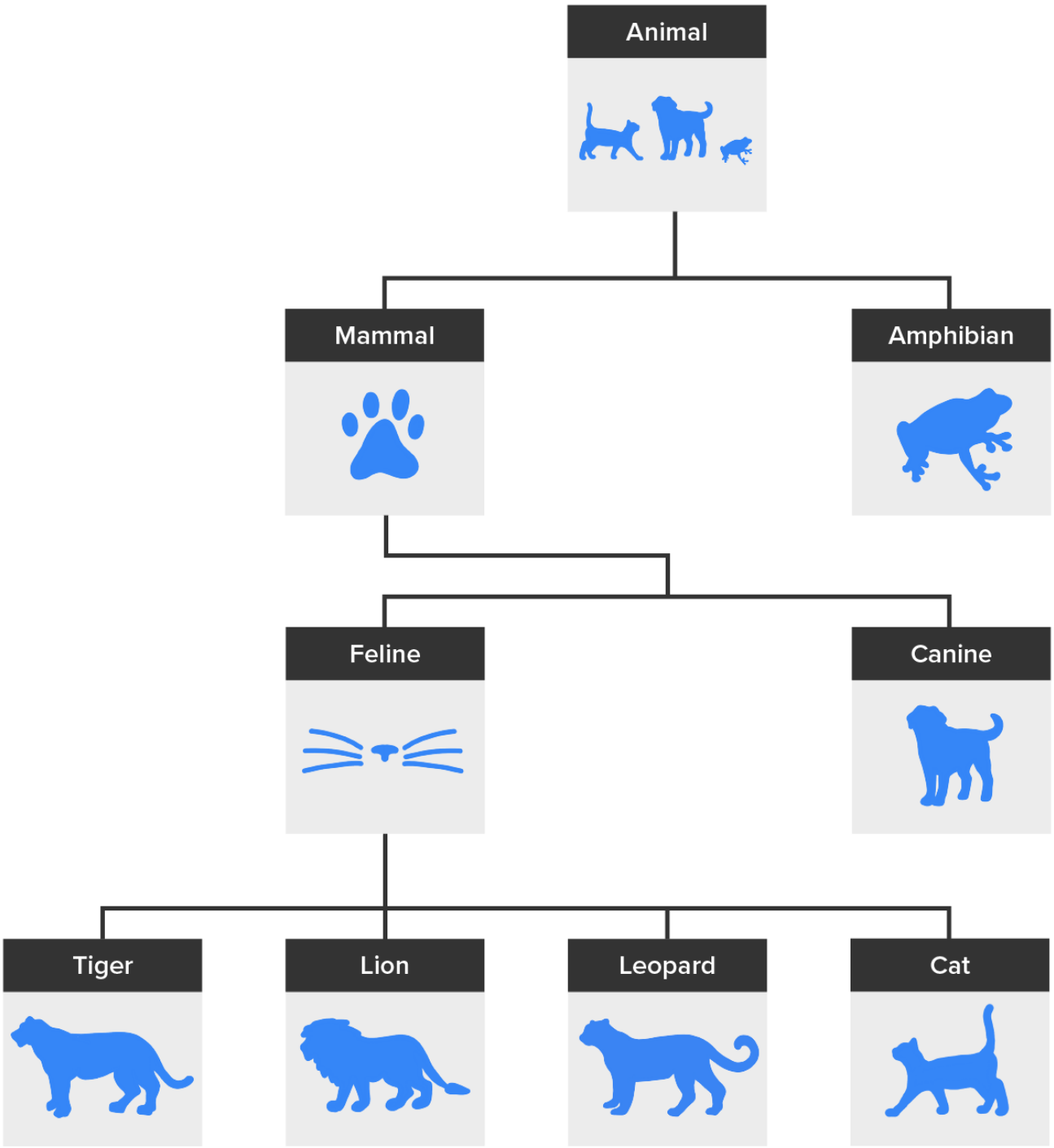
Looking at the graphic above, Persian, Siamese, and British Shorthair are unique breeds of cats, but what makes cats similar to one another are the characteristics that they all inherit from the class `Cat`.

Classes and class inheritance were created to align with much of the information in the real world and how it can be stored, categorized, and understood by using classes and subclasses. For example, you may have noticed other creatures similar to cats that are in the real world. They would be similar to cats in that they inherit features from a higher-level class that we could call Felines.



You could have the class Feline, and under it, you could have Tiger, Leopard, Lion, and Cat as subclasses.

Above Feline, you could have Mammals as a layer, as all felines are mammals. Then, on top of Mammals, you could have a layer called Animals since all mammals are animals.



This concept doesn't just apply to cats. It can apply to other animals like dogs or amphibians. If you search online to see the hierarchy of animals, you'll see that there are many ways to classify animals, and also see how inheritance works all the way down various lines to specific animals.

From a coding perspective, the easiest way to use inheritance is to create "child" classes (or subclasses) from "parent" classes or (base classes).

IN CONTEXT

You have your mother's eyes, and you have the same patience as your father...? These are things that are relevant to the term “inheritance.” You inherited those objects or traits without having to do anything.

KEY CONCEPT

In the world of Java programming (and programming more generally), inheritance means the same thing—a “child” class inheriting characteristics (data) and behaviors (methods) from a “parent” class without modifying anything.

You will hear a few terms in this Challenge that may be defined differently in other resources you may be using, so it makes sense to call them out.

Inheritance is a relationship model that allows us to define a class that inherits all the attributes and methods from another class.

A **base class** is also known as a parent class. It is a class that is being inherited from.

A **subclass** is also known as a child class. It is a class that inherits from another class. A subclass can “extend” the base class, meaning that it can define additional data that will not affect the base class.

From here on, we will be using the terms "base class" and "subclass."

A base class defines all of the things that apply to all instances of that class. Therefore, a subclass automatically inherits what was defined in the base class. Once the subclass is created, though, any data/methods defined in that subclass are relevant only to that subclass. Anything that is defined in the subclass will not change or replace anything that is coming from the base class. These definitions will also not affect any other subclasses of the base class.

BIG IDEA

So, what is the benefit of inheritance in Java? Inheritance allows programmers to create subclasses that have the properties and methods of an existing class. This supports code reusability, maintenance, and optimization; and it speeds up development time. You will not have to “reinvent the wheel” with each class. You can create it once and use it multiple times. Back to our feline example. If you have a base class with all the common attributes and standard behaviors (methods) that every feline should have, you should never need to alter that base class when creating subclasses. You inherit those properties and methods automatically, so there is no need to define those again. Also, if something changes in the base class, once that change is made, it automatically cascades those changes to each subclass.

TERMS TO KNOW

Base Class

A base class is also known as a parent class or superclass. It is a class that is being inherited from.

Subclass
A subclass is also known as a child class. It is a class that inherits from another class. A subclass can “extend” the base class, meaning that it can define additional data and related behavior that will not affect the base class.

Inheritance
Inheritance is a relationship model that allows us to define a class that inherits all the attributes and methods from another class.

---

2. Base Class

Subclasses inherit the public methods (behaviors/actions) of a higher-level class or the base class. A base class is no different than any other class, but it is simply a class that a subclass would inherit from.

To help define these inheritance concepts and start reviewing code, let’s start with a class called `Member`:

TRY IT

Directions: Input the following into 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);
  }

  public String getFirstName() {
    return firstName;
  }

  public String getLastName() {
    return lastName;
  }
```

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

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

REFLECT

In the code, you may find it quite familiar, as we used something very similar in a prior tutorial. We are importing the `library class java.time.LocalDate` to use the current date functionality. Again, you will be seeing module importation and usage in the next challenge. Then, we defined our class called `Member`. There is an attribute declared called `expiryDays`, and we set it to "365". It is used to calculate the expiration date from the current date. We then defined the constructor with three parameters. The first is the “self” parameter, which is used to point to the instance being referenced. Next are the “first” and “last” parameters, which are then assigned to the variables `first_name` and `last_name`. The `expiry_date` variable is set to "365" days after today when we run the code. The `.timedelta()` method (included in the `datetime` module) calculates the difference of time between the two arguments.

Next, we will create an instance of this class with the variable `TestMember` and create some `print()` functions so we can see some output to the screen.

Let’s give this a test.

TRY IT

Directions: Create a `MemberProgram` class in a file named MemberProgram.java:

```
class MemberProgram {
  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());
  }
}
```

TRY IT

Directions: Run the `MemberProgram`:

The output table below is the result of this small program:

```
First Name: Sophia
```

```
Last Name:  Java
Exp. Date: 2025-05-07
```

THINK ABOUT IT

See anything new? Not really. This is very similar to a class we built out in the last challenge. You have already created this class to use as a base class for the examples to follow. Now, the intent is to create two different kinds of subclasses of the Member class.

In the next topic, you will create an `Admin` and a `User` subclass. Both of these subclasses will have attributes that the `Member` class includes. These new classes will be subclasses of our `Member` class. They will automatically include the attributes and methods of our base class (`Member`).

When you define those types of members as a subclass of `Member`, they will automatically be assigned the same attributes and methods if there are any defined.

---

3. Subclasses

To create and define a subclass, you will need to ensure that the subclass is below the base class and is not indented. The subclass is not part of or contained within the base class.

Therefore, you would use the following syntax:

```
class SubclassName extends BaseClassName {
```

It is important to note the use of the keyword `extends` between the name of the subclass and the base class.

You would replace `SubclassName` with what you want to name the subclass. Then, you would replace `BaseClassName` with the name of the base class. For example, to make a subclass of our base class `Member` and name it `Admin`, you would create a file named Admin.java and start with the following code:

```
public class Admin extends Member {
```

The code for the first version of this class should look like this:

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

At this point, you would keep the content and functionality of the subclasses simple. They don't have any attributes or methods of their own, other than a parameterized constructor. This simple structure will allow you to test the two subclasses.

You can do the same thing for the `User` class as well.

TRY IT

Directions: The following code should be entered in a file named User.java:

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

To do a quick test to ensure that this works, you would create an `Admin` and a `User` instance as subclasses that will simply inherit all of the methods from our base class (`Member`). You would also include some output statements to view and evaluate the logic of the program so far.

TRY IT

Directions: Save the following code in a file named MemberInheritance.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());
  }
}
```

We added some different arguments to the subclass instances besides “Sophia” and “Java,” so the output will be clearer. Again, using the same output statements for each of the instances, the results of this would look like

the following:

```
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
```

REFLECT

Here we see the outputs for the base class (`Member`) and both instances of the subclasses (`Admin` and `User`). At this point, all three classes produce the same output, but note that all of the attributes and the methods for accessing them are in the `Member` base class. The two subclasses just have parameterized constructors that call the superclass's constructor (via the `super()` constructor).

BRAINSTORM

Directions: Build out a base class and create a subclass. Practice on some of the code above or create your own.

SUMMARY

In this lesson, you learned about inheritance by looking at **real-world examples**. The use of inheritance and classes in a programming language increases the efficiency and supports the “build once and use many times” analogy. You saw that any defined class can be the **base class** whose attributes (properties) and methods (behaviors/actions) can be inherited from. **Subclasses** are defined as the classes that inherit from the base class. A subclass can “extend” the base class, meaning that it can define additional data that will not affect the base class. Finally, you saw the output of two subclass instances inherit attributes from a base class that we built.

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

Base Class

A base class is also known as a parent class or superclass. It is a class that is being inherited from.

Inheritance

Inheritance is a relationship model that allows us to define a class that inherits all the attributes and methods from another class.

Subclass

A subclass is also known as a child class. It is the class that inherits from another class. A subclass can “extend” the base class, meaning that it can define additional data and related behavior that will not affect the base class.