**Johns Hopkins University**
**Introduction to Programming Using Java**
**605.201**

# Introduction to Object-Oriented Concepts

Learning object-oriented programming (OOP) is both simple and complex. It's complex because it requires a student to mentally shift gears and develop a new way of thinking about writing a computer program…and this is not an easy task. So far in this course you have learned basic computing concepts and the basic elements of the Java language, but most of the programs you have designed didn't incorporate object-oriented concepts. That's all going to change from this point on. On the other hand, while learning OOP can be complex it is also simple in the sense that many people already think naturally in terms of objects.

As an example of how natural it is to think in terms of objects, I'd like you to think about a pet that you might have or that a friend might have…maybe a cat or a dog. When thinking about a cat or a dog it is natural for most people to see their pet as an object. A pet has attributes, such as weight, color, and age. And, a pet also has behavior…it eats, it barks or meows, and it walks.

Come to think of it, a person can be thought of as an object. A person has a weight, a height, an age, is right-handed or left-handed, and so forth. And a person certainly has behavior…a person may walk, talk, run, etc.

So…you may be wondering how this all relates to OOP. Well, the definition of an **object** in OOP is something that can be described by both data attributes (e.g., age, weight) and behavior (e.g., eat, talk). And that something can be a model of a real-world thing…like a pet…or a more abstract concept…like the protocol for sending information to and from a web browser over the Internet.

# Johns Hopkins University
**Introduction to Programming Using Java**
**605.201**

## Introduction to Object-Oriented Concepts

In OOP we begin solving problems by identifying relevant objects as well as the attributes and behaviors of those objects. This is called object-oriented design (OOD) and it's a very important and necessary step that precedes the programming step.

Let's take an example to see how we can perform OOD to model a cat. We begin the design process by deciding that we need to design a cat. Then, we identify the relevant attributes and behavior needed to model a cat. For discussion purposes we'll keep it simple. Let's use the attributes below to describe a cat.

**<u>Cat Attributes</u>**

Breed
Age
Weight
Sex
Name

And, let's assume we're only interested in two behaviors for our example:

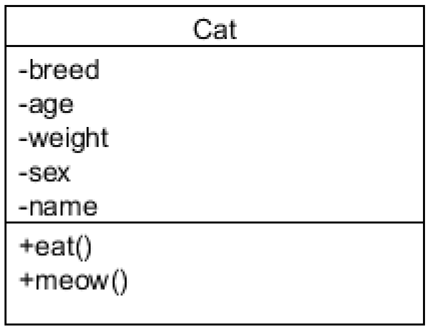
**<u>Cat Behaviors</u>**

Eating
Meowing

To document our design we will use a special kind of diagram called a class diagram, which will look like this for the cat example:

**A Simple Class Diagram**

## Introduction to Object-Oriented Concepts



The class diagram is very simple, but it documents our cat design concisely. The diagram uses a rectangle as an icon for our model of a cat. The rectangle contains three compartments. The top compartment contains the name of the thing being modeled…in our case a cat. The next compartment contains a list of the attributes we are including in our model. And the bottom compartment contains a list of relevant behaviors. Note that parentheses were added to the behavior names to indicate that they will correspond to methods when we actually write the code.

I'm sure you noticed the use of the hyphens (-) and plus signs (+) in the diagram. For now, don't worry what they mean. You'll learn more about them later in this course unit. You probably also noticed that cat is referred to as a class name…and not an object name as you might have expected. So, what's up with that? Well, here's where things may get a wee bit complicated.

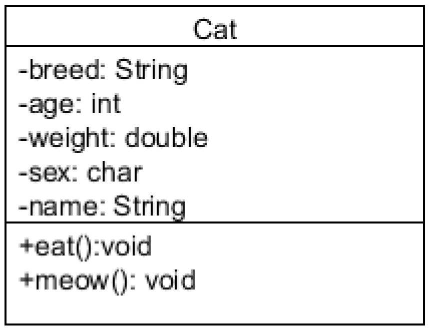
OOP languages contain a programming construct called a **class**, that we use to help create objects. A class is like a blueprint that a programming language uses to create an object. In OOP all objects must be created from a class definition. The class is a very powerful concept because it saves us a lot of programming work. Once we have a class definition for modeling a cat we can create many cat objects in our programs without doing much programming.

## Introduction to Object-Oriented Concepts

At this point you may be wondering how we actually implement the descriptive attributes and behaviors in code. The attributes are specified using types…such as primitive or non-primitive Java types that you've learned about. And the behaviors are implemented using methods.

Now, let's take our basic cat design and refine it a bit. Knowing what you do about the Java language, what types do you think would be appropriate for the attributes of a cat? The types can be primitive types or non-primitive types…and there can exist a variety of choices that we can make as designers. Let's start with the age attribute. A good type for that attribute would be an integer since we commonly think of ages as being described in integer years. For the weight attribute, let's use a double so that we can keep track of a cat's weight more accurately…a whole pound can be a lot of weight difference for a small cat. For the sex, let's use a character type…'f' for female and 'm' for male. And for the breed and name attributes let's use a String type. Our refined model is as follows:

**A Refined Class Diagram**



This refined diagram tells us exactly what types to use for each attribute when we ultimately write the code for our Cat class. Now, you may have noticed that I added void to the behaviors compartment in our refined model. The eat and meow behaviors will be implemented as methods when we write the code, and I'm just specifying here that they will have a void return type.

And now…the moment you've been eagerly anticipating. Let's see how we can turn this design into code, by providing a class definition.

```
public class Cat
{
        private String breed;
        private int age;
        private double weight;
        private char sex;
        private String name;

        public void eat()
        {
                System.out.println( "Munch, Munch, Munch" );
        }

        public void meow()
        {
```

## Introduction to Object-Oriented Concepts

```
                System.out.println( “Meeeeeooww” );
        }
}
```

There it is…a complete class definition written in the Java programming language. Note that to keep things simple I just displayed some basic information for the eat and meow behaviors. Oh, and don’t worry if you don’t understand what public and private mean and why I didn’t use a main() method and stuff like that. You’ll learn about all the nuances in this course unit. For now, just go with understanding the concepts that have been introduced.

Now, let’s write a program to create some cats. To do that I’m going to define a class called CreateCats that does all the work…and the code is going to look somewhat familiar to you.

```
public class CreateCats
{
        public static void main( String [] args )
        {
                // Create two Cat objects
                Cat catOne = new Cat();
                Cat catTwo = new Cat();

                // Make the cats meow
                System.out.print( “cat 1 says “ ); catOne.meow();
                System.out.print( “cat 2 says “ ); catTwo.meow();
        }
}
```

The above program creates two Cat objects and makes them meow by invoking their meow behavior. Creating the Cat objects should look familiar to you…we use the **new** operator and it’s very similar to how one would create a String object that you learned about in an earlier course unit. Similarly, invoking the meow() method is similar to invoking any of the methods in the Java String class. The variables catOne and catTwo are simply references to the two Cat objects and are used to manipulate each cat.

Much of the work that is done in OOP is designing and implementing classes, then connecting those classes to work together to get stuff done. Technically, OOP could be called class-oriented programming…but then it wouldn’t have quite the same ring to it.

This brief tutorial is intended to provide a quick introduction to object-oriented thinking and to illustrate how to design and write code that implements a simple class. You’ll learn a lot more about the technicalities as you work through the course material.