Catanduanes State University

College of Information and Communications Technology

Virac, Catanduanes

Image from: https://www.freecodecamp.org/news/why-we-will-always-need-new-programming-languages-3415869ea37e/

LEARNING MATERIALS AND COMPILATION OF LECTURES/ACTIVITIES

ITRACKC1

Integrative Programming and Technologies 2: Object-Oriented Programming

TABLE OF CONTENTS

CHAPTER 1: BASIC CONCEPT OF OBJECT-ORIENTED PROGRAMMING 1
LEARNING OUTCOMES 2
KEY TERMS 2
LESSON 1: CONCEPT OF OOPS 2
1.1. OBJECT-ORIENTED PROGRAMMING IN JAVA 2
1.2. THE TWO PROGRAMMING PARADIGMS 3
1.3. ABSTRACTION 3
1.4. ENCAPSULATION 3
1.5. INHERITANCE 5
1.6. POLYMORPHISM 5
LESSON 2: BENEFITS OF OOPS 6
2.1. BENEFITS OF OBJECT-ORIENTED PROGRAMMING 7
LESSON 3: OBJECTS VS CLASSES 8
3.1. CLASS FUNDAMENTALS 8
3.2. THE GENERAL FORM OF A CLASS 9
REFERENCES 10

CHAPTER 1: BASIC CONCEPT OF OBJECT-ORIENTED PROGRAMMING (OOP)

LEARNING OUTCOMES

At the end of this chapter, the students shall be able to:

1. Know basic concepts of object-oriented programming.
2. Define benefits of object-oriented programming.

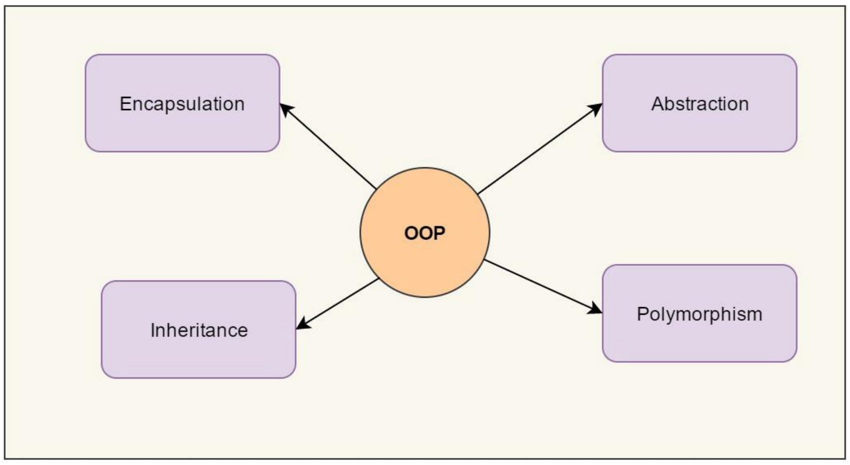
KEY TERMS

1. Object-Oriented Programming
2. Process-Oriented Model
3. Objects
4. Classes
5. Inheritance
6. Encapsulation
7. Abstraction
8. Polymorphism
9. Hierarchical Abstraction
10. Methods
11. Functions
12. Java

LESSON 1: CONCEPT OF OOPS

Object-Oriented Programming is a programming style which is associated with the concepts like class, object, inheritance, encapsulation, abstraction, and polymorphism. Most popular programming languages like Java, C++, C#, Ruby, etc. follow an object-oriented programming paradigm.

1.1. OBJECT-ORIENTED PROGRAMMING IN JAVA



Four Pillars of Object Oriented Programming

Image from: https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727

Object-Oriented programming (OOP) refers to a type of programming in which programmers define the data type of a data structure and the type of operations that can be applied to the data structure. As Java being the most sought-after skill, we will talk about object-oriented programming concepts in Java. An object-based application in Java is based on declaring classes, creating objects from them and interacting between these objects.

OOP is at the core of Java. In fact, all Java programs are to at least some extent object-oriented. OOP is so integral to Java that it is best to understand its basic principles before you begin writing even simple Java programs. Therefore, this chapter begins with a discussion of the theoretical aspects of OOP.

1.2. THE TWO PROGRAMMING PARADIGMS

All computer programs consist of two elements: code and data. Furthermore, a program can be conceptually organized around its code or around its data. That is, some programs are written around "what is happening" and others are written around "who is being affected." These are the two paradigms that govern how a program is constructed:

1. **Process-Oriented Model** – This approach characterizes a program as a series of linear steps (that is, code). The process-oriented model can be thought of as code acting on data. Procedural languages such as C employ this model to considerable success. However, problems with this approach appear as programs grow larger and more complex.

2. **Object-Oriented Programming** – Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code. As you will see, by switching the controlling entity to data, you can achieve several organizational benefits.

1.3. ABSTRACTION

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. This abstraction allows people to use a car to drive to the grocery store without being overwhelmed by the complexity of the individual parts. They can ignore the details of how the engine, transmission, and braking systems work. Instead, they are free to utilize the object as a whole.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system might consist of a radio, a CD player, and/or MP3 player. The point is that you manage the complexity of the car (or any other complex system) through the use of hierarchical abstractions.

Hierarchical abstractions of complex systems can also be applied to computer programs. The data from a traditional process-oriented program can be transformed by abstraction into its component objects. A sequence of process steps can become a collection of messages between these objects. Thus, each of these objects describes its own unique behavior. You can treat these objects as concrete entities that respond to messages telling them to do something. This is the essence of object-oriented programming.

1.4. ENCAPSULATION

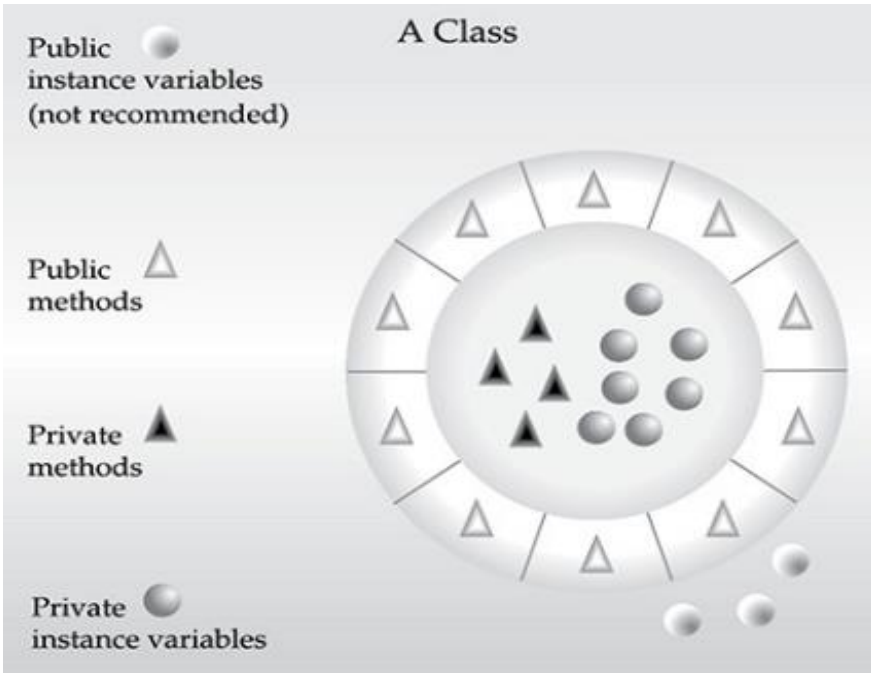
Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about your engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever. You, as the user, have only one method of affecting this complex encapsulation: by moving the gear-shift lever. You can't affect the transmission by using the turn signal or windshield wipers, for example. Thus, the gear-shift lever is a well-defined (indeed, unique) interface to the transmission. Further, what occurs inside the transmission does not affect objects outside the transmission. For example, shifting gears does not turn on the headlights! Because an automatic transmission is encapsulated, dozens of car manufacturers can implement one in any way they please. However, from the driver's point of view, they all work the same. This same idea can be applied to programming. The power of encapsulated code is that everyone knows how to access it and thus can use it regardless of the implementation details—and without fear of unexpected side effects.

In Java, the basis of encapsulation is the class. A class defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality.

When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods. (If you are familiar with C/C++, it may help to know that what a Java programmer calls a method, a C/C++ programmer calls a function.) In properly written Java programs, the methods define how the member variables can be used. This means that the behavior and interface of a class are defined by the methods that operate on its instance data. Objects and classes will be furtherly discussed in the next lessons/chapters.

Since the purpose of a class is to encapsulate complexity, there are mechanisms for hiding the complexity of the implementation inside the class. Each method or variable in a class may be marked private or public. The public interface of a class represents everything that external users of the class need to know, or may know. The private methods and data can only be accessed by code that is a member of the class. Therefore, any other code that is not a member of the class cannot access a private method or variable. Since the private members of a class may only be accessed by other parts of your program through the class' public methods, you can ensure that no improper actions take place. Of course, this means that the public interface should be carefully designed not to expose too much of the inner workings of a class.

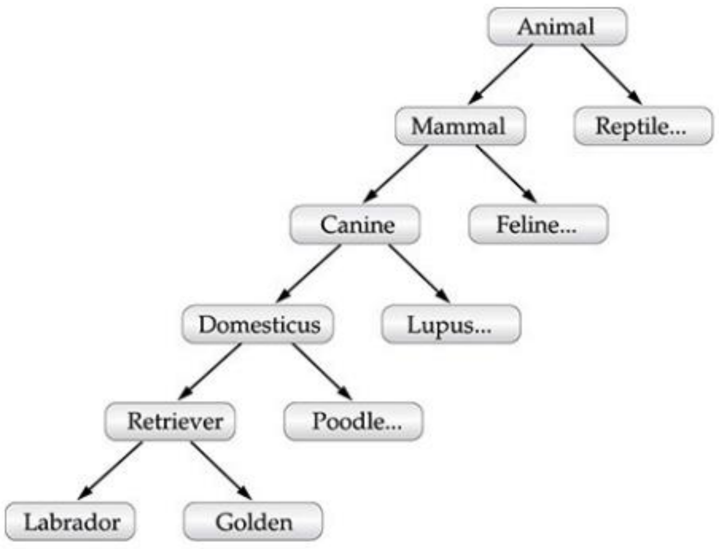

A Class
Public instance variables (not recommended)
Public methods
Private methods
Private instance variables

1.5. INHERITANCE

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Most people naturally view the world as made up of objects that are related to each other in a hierarchical way, such as animals, mammals, and dogs. If you wanted to describe animals in an abstract way, you would say they have some attributes, such as size, intelligence, and type of skeletal system. Animals also have certain behavioral aspects; they eat, breathe, and sleep. This description of attributes and behavior is the class definition for animals.

If you wanted to describe a more specific class of animals, such as mammals, they would have more specific attributes, such as type of teeth and mammary glands. This is known as a subclass of animals, where animals are referred to as mammals' superclass. Since mammals are simply more precisely specified animals, they inherit all of the attributes from animals. A deeply inherited subclass inherits all of the attributes from each of its ancestors in the class hierarchy.



1.6. POLYMORPHISM

Polymorphism (from Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non–object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action (that is, method) as it applies to each situation. You, the programmer, do not need to make this selection manually. You need only remember and utilize the general interface.

Extending the dog analogy, a dog’s sense of smell is polymorphic. If the dog smells a cat, it will bark and run after it. If the dog smells its food, it will salivate and run to its bowl. The same sense of smell is at work in both situations. The difference is what is being smelled, that is, the type of data being operated upon by the dog’s nose. This same general concept can be implemented in Java as it applies to methods within a Java program.

SUPPLEMENTARY LEARNING RESOURCES

Schildt, H. (2019). *Java: The Complete Reference Eleventh Edition*. Oracle Press.

SELF-ASSESSMENT QUESTION 1

Differentiate the four (4) principles of OOP.

LESSON 2: BENEFITS OF OOPS

OOP has become a fundamental part of software development. Thanks to the ubiquity of languages like Java and C++, you can’t develop software for mobile unless you understand the object-oriented approach. The same goes for serious web development, given the popularity of OOP languages like Python, PHP and Ruby. Getting your head around the idea of object-oriented programming can be challenging for some IT professionals. You may be wondering why you even need objects when you could use the top-down approach of traditional structured programming in languages like Visual Basic.

If you’ve written this kind of software, you’re probably used to breaking down large problems into sub-problems and solving them in separate units of code. Or you may have experience with functional programming, which treats elements of code as precise mathematical functions, and prevents them from affecting other elements, i.e., no side effects.

2.1. BENEFITS OF OBJECT-ORIENTED PROGRAMMING

Here's a detailed look at some of OOP's top benefits:

1. MODULARITY FOR EASY TROUBLESHOOTING

   Something has gone wrong, and you have no idea where to look. Will you have to trudge through that code file? Hope you commented your code. When working with object-oriented programming languages, you know exactly where to look. "Oh, the car object broke down? The problem must be in the Car class!" You don't have to muck through anything else.

   That's the beauty of encapsulation. Objects are self-contained, and each bit of functionality does its own thing while leaving the other bits alone. Also, this modality allows an IT team to work on multiple objects simultaneously while minimizing the chance that one person might duplicate someone else's functionality.

2. REUSE OF CODE THROUGH INHERITANCE

   Suppose that in addition to your Car object, one colleague needs a RaceCar object, and another needs a Limousine object. Everyone builds their objects separately but discover commonalities between them. In fact, each object is really just a different kind of Car. This is where the inheritance technique saves time: Create one generic class (Car), and then define the subclasses (RaceCar and Limousine) that are to inherit the generic class's traits.

   Of course, Limousine and RaceCar still have their unique attributes and functions. If the RaceCar object needs a method to "fireAfterBurners" and the Limousine object requires a Chauffeur, each class could implement separate functions just for itself. However, because both classes inherit key aspects from the Car class, for example the "drive" or "fillUpGas" methods, your inheriting classes can simply reuse existing code instead of writing these functions all over again.

   What if you want to make a change to all Car objects, regardless of type? This is another advantage of the OO approach. Simply make a change to your Car class, and all car objects will simply inherit the new code.

3. FLEXIBILTY THROUGH POLYMORPHISM

   Riffing on this example, you now need just a few drivers, or functions, like "driveCar," "driveRaceCar" and "DriveLimousine." RaceCarDrivers share some traits with LimousineDrivers, but other things, like RaceHelmets and BeverageSponsorships, are unique.

   This is where object-oriented programming's sweet polymorphism comes into play. Because a single function can shape-shift to adapt to whichever class it's in, you could create one function in the parent Car class called "drive" — not "driveCar" or "driveRaceCar," but just "drive." This one function would work with the RaceCarDriver, LimousineDriver, etc. In fact, you could even have "raceCar.drive(myRaceCarDriver)" or "limo.drive(myChauffeur)."

4. EFFECTIVE PROBLEM SOLVING

   A language like C has an amazing legacy in programming history, but writing software in a top-down language is like playing Jenga while wearing mittens. The more complex it gets, the greater the chance it will collapse. Meanwhile, writing a functional-style program in a language like Haskell or ML can be a chore.

   Object-oriented programming is often the most natural and pragmatic approach, once you get the hang of it. OOP languages allows you to break down your software into bite-sized problems that you then can solve — one object at a time.

   This isn't to say that OOP is the One True Way. However, the advantages of object-oriented programming are many. When you need to solve complex programming challenges and want to add code tools to your skill set, OOP is your friend — and has much greater longevity and utility than Pac-Man or parachute pants.

SUPPLEMENTARY LEARNING RESOURCES

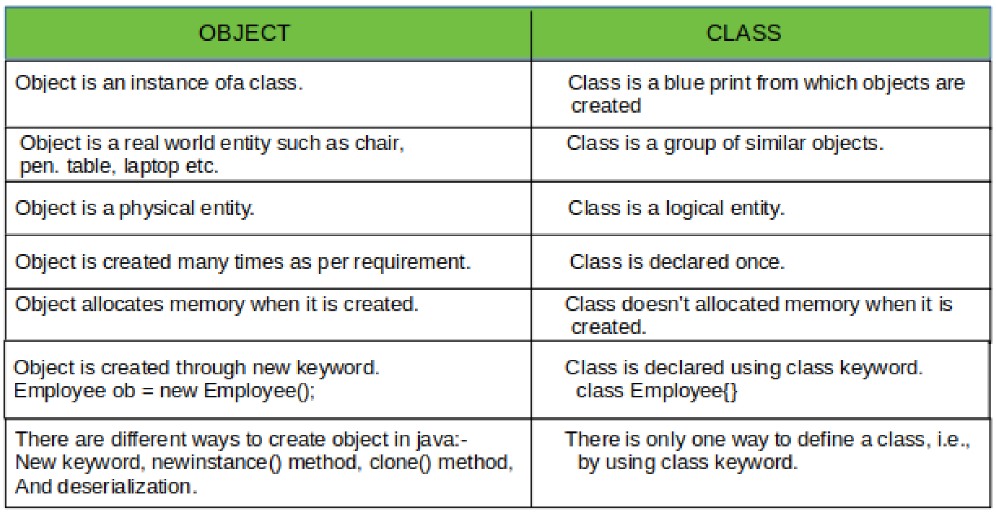
Half, R. (2017). *4 Advantages of Object-Oriented Programming*. Retrieved from https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming

SELF-ASSESSMENT QUESTION 2

Based from what you have learned so far, add some ideas on what other benefits OOP can offer. List at least 3 benefits.

LESSON 3: OBJECTS VS CLASSES

The class is at the core of Java. It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object. As such, the class forms the basis for object-oriented programming in Java. Any concept you wish to implement in a Java program must be encapsulated within a class. As said in Lesson 2, study the table below.



OBJECT	CLASS
Object is an instance ofa class.	Class is a blue print from which objects are created
Object is a real world entity such as chair, pen. table, laptop etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
Object is created through new keyword. Employee ob = new Employee();	Class is declared using class keyword. class Employee{}
There are different ways to create object in java:- New keyword, newinstance() method, clone() method, And deserialization.	There is only one way to define a class, i.e., by using class keyword.

Image from: https://www.geeksforgeeks.org/understanding-classes-and-objects-in-java/

3.1. CLASS FUNDAMENTALS

The classes created in the preceding lesson primarily exist simply to encapsulate the `main()` method, which has been used to demonstrate the basics of the Java syntax. As you will see, classes are substantially more powerful than the limited ones presented so far. Perhaps the most important thing to understand about a class is that it defines a new data type. Once defined, this new type can be used to create objects of that type. Thus, a class is a template for an object, and an object is an instance of a class. Because an object is an instance of a class, you will often see the two words object and instance used interchangeably.

3.2. THE GENERAL FORM OF A CLASS

When you define a class, you declare its exact form and nature. You do this by specifying the data that it contains and the code that operates on that data. While very simple classes may contain only code or only data, most real-world classes contain both. As you will see, a class' code defines the interface to its data.

A class is declared by use of the **class** keyword. The classes that have been used up to this point are actually very limited examples of its complete form. Classes can (and usually do) get much more complex. A simplified general form of a class definition is shown here:

```
class ClassName {
      datatype instanceVariable1;
      datatype instanceVariable2;
      // …
      datatype instanceVariableN;

      type methodName1(parameterList) {
            //body of method
      }

      type methodName2(parameterList) {
            //body of method
      }

      // …

      type methodNameN(parameterList) {
            //body of method
      }
}
```

The data, or variables, defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class. In most classes, the instance variables are acted upon and accessed by the methods defined for that class. Thus, as a general rule, it is the methods that determine how a class' data can be used.

Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another. We will come back to this point shortly, but it is an important concept to learn early.

All methods have the same general form as **main()**, which we have been using thus far. However, most methods will not be specified as static or public. Notice that the general form of a class does not specify a **main()** method. Java classes do not need to have a **main()** method. You only specify one if that class is the starting point for your program. Further, some kinds of Java applications don't require a **main()** method at all.

SUPPLEMENTARY LEARNING RESOURCES

Schildt, H. (2019). *Java: The Complete Reference Eleventh Edition*. Oracle Press.

SELF-ASSESSMENT QUESTION 3

Write your own ideas on how objects relate to classes.

REFERENCES

Half, R. (2017). *4 Advantages of Object-Oriented Programming*. Retrieved from https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming.

Schildt, H. (2019). *Java: The Complete Reference Eleventh Edition*. Oracle Press.