UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING
Computer Engineering
2nd Semester, School Year 2024-2025

Object-Oriented Programming

Laboratory Activity No. 1

Review of Technologies

Submitted by:
Balana, Jerkielle Roen O.
Sat. 4:30PM-8:30PM / BSCpE 1A

Submitted to
Ma'am Sayo
Professor

Date Performed:
18/01/2025

Date Submitted
18/01/2025

I. Objectives

In this section, the goals in this laboratory are:

- To define the key terms in Object-oriented programming
- To be able to know the construction of OO concepts in relation to other types of programming such as procedural or functional programming

II. Methods

General Instruction:

A. Define and discuss the following Object-oriented programming concepts:

1. Classes

Programs use classes as parameters to help them generate objects. Not all OOP coding languages have classes, even if they all allow objects. Both classes and objects are supported by many coding languages. Individual objects have distinct data for every qualification, and each class has its own set of requirements. It stands for the collection of attributes or functions shared by all objects of a particular type. A class functions similarly to an object's blueprint. The structure that specifies the data and the methods to manipulate it is called a class. Whether you create a class yourself or use one from the Java platform API libraries, all program data is encapsulated in a class while writing Java applications.

2. Objects

This is the fundamental component of object-oriented programming. In other words, an object is a collection of data and functions that work with data. It is an essential component of Object-Oriented Programming and represents real-world entities. An instance of a class is called an object. Memory is allocated when a class is instantiated, or when an object is formed, but not when it is defined. Every object has a state, identity, and behavior. Data and code to work with the data are contained in every object. It is necessary to know the sort of message that the objects accept and the type of response that they provide in order for them to interact; they do not need to know specifics about each other's data or code.

3. Fields

An object's state or properties are represented by fields, often known as instance variables. They enable the distinction between instances of the same class by storing information unique to each object. In order to preserve data integrity and guarantee that objects perform as intended, fields must be managed properly.

4. Methods

Functions and subroutines known as methods specify how a class or its objects should behave. Methods usually determine how an object may utilize or alter data and are specific to a class or object. When an object is a member of a class, the class's methods may take precedence over the object's methods, altering or improving how the class's objects use data or react to commands.

5. Properties

Properties provide for more controlled access to and modification of fields. To improve data security and encourage encapsulation, developers can use properties to add validation logic or encapsulate field access. This characteristic aids in preserving a distinct interface between an object's exterior interactions and internal state. Also,
an object-oriented idiom is properties. A 'getter' that retrieves a value and a'setter' that sets a value are the two functions that are described by the phrase, depending on the desired program behavior. Properties typically don't have many negative impacts. (And their side effects are usually limited: they can convert an object's private data to or from a publicly-declared type, alert listeners of a change, or validate the item being set.)

III. Results

Classes are fundamental constructs in object-oriented programming (OOP) that serve as blueprints for creating objects. They encapsulate data for the object and methods to manipulate that data, enabling organized and efficient program development.

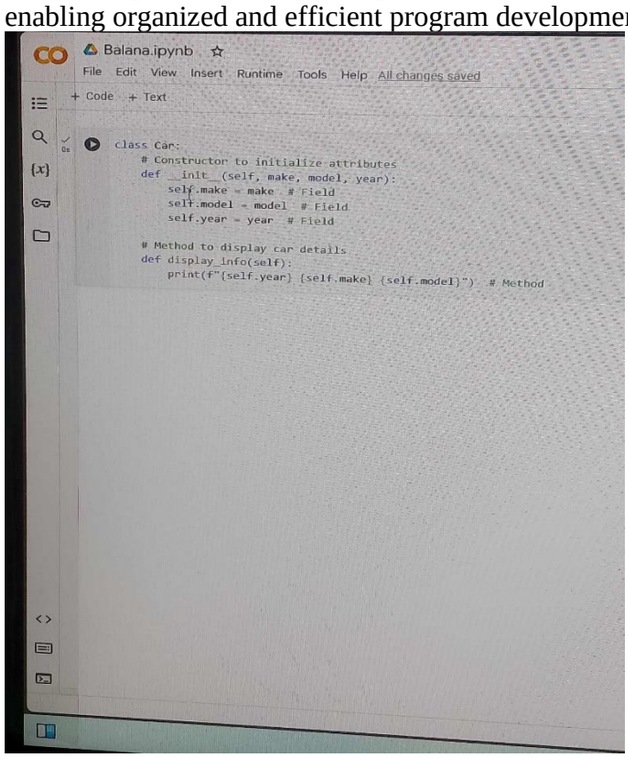


Figure 1. The figure represents that the class is a blueprint for creating objects.

Example:Car.

Objects are instances of classes that stand in for tangible things or abstract ideas. They are the main parts of a software system that uses object-oriented programming. Objects have related methods that specify their behavior and store data in the form of attributes.

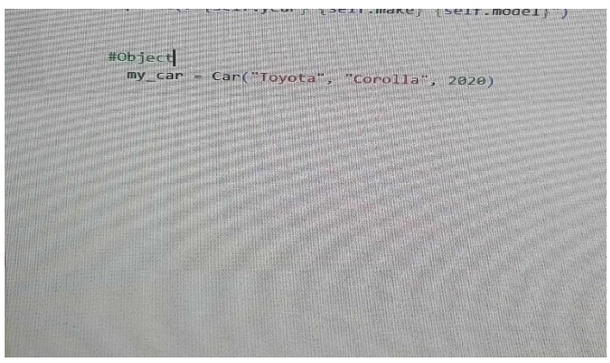


Figure 2. The object is an instance of a class that represents a specific entity. Example: my car (an instance of Car).

Fields represent the data or state of an object. They store information about an object's characteristics, such as its color, size, or name. Fields are typically defined within a class and are assigned values when an object is instantiated.

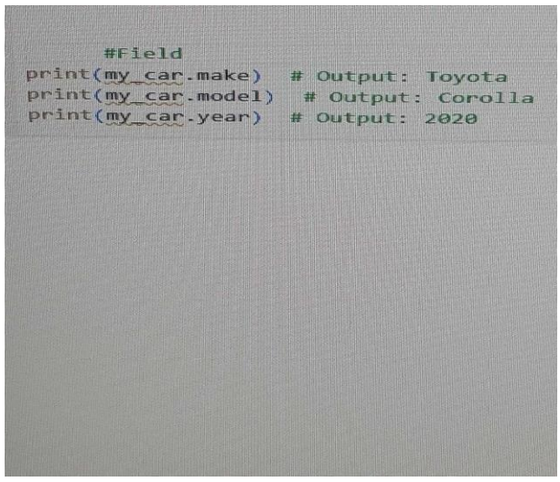


Figure 3. Variables that hold data related to the object. Example:make, model, year in the car class.

Methods are functions that exist within a class and define the behavior of its objects. They interact with other things, carry out operations, and modify data. Developers can design large systems with a high degree of order and modularity by using methods to enable communication between components. Methods often change, update, or remove data. However, they don't always need to modify data. Methods help programmers encourage reusability and keep features contained within an object. This makes debugging easier, as errors can be found and fixed in a single location.

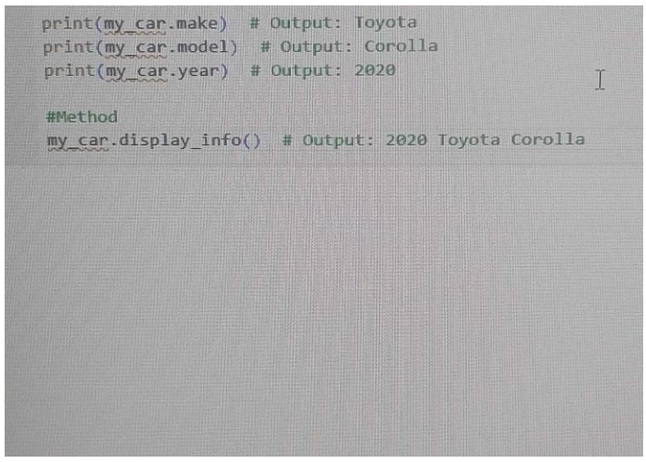
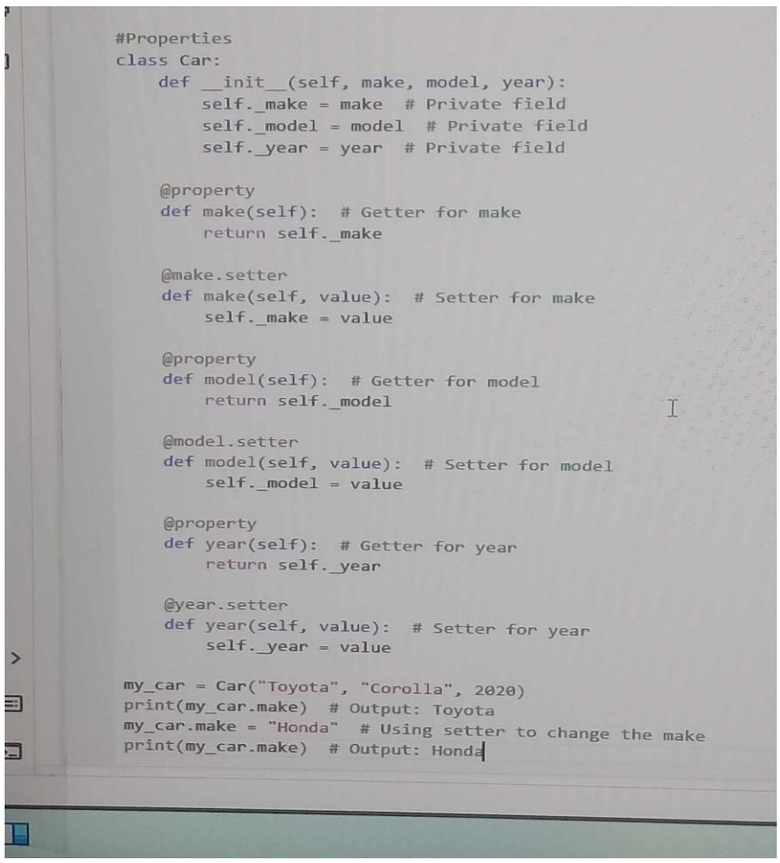


Figure 4. Functions defined within a class that describe the behaviors of an object. Example: display_info()

Properties are an important component in Python's object-oriented programming (OOP) framework that enable regulated access to instance attributes. They give you a means to control an object's internal state while preserving encapsulation by allowing you to specify methods that are accessible, much like attributes. A property in Python is a special kind of attribute that allows you to define getter, setter, and deleter methods for managing access to an attribute. This is done using the @property decorator, which makes it possible to use methods as if they were attributes.



```
#Properties
class Car:
    def __init__(self, make, model, year):
        self._make = make  # Private field
        self._model = model  # Private field
        self._year = year  # Private field

    @property
    def make(self):  # Getter for make
        return self._make

    @make.setter
    def make(self, value):  # Setter for make
        self._make = value

    @property
    def model(self):  # Getter for model
        return self._model

    @model.setter
    def model(self, value):  # Setter for model
        self._model = value

    @property
    def year(self):  # Getter for year
        return self._year

    @year.setter
    def year(self, value):  # Setter for year
        self._year = value

my_car = Car("Toyota", "Corolla", 2020)
print(my_car.make)  # Output: Toyota
my_car.make = "Honda"  # Using setter to change the make
print(my_car.make)  # Output: Honda
```

Figure 5. Special methods (getters and setters) that provide controlled access to the fields of an object. Example: Using properties to manage access to fields.

IV. Conclusion

We have studied the core ideas of object-oriented programming (OOP) in this lab activity, with particular attention to classes, objects, methods, fields, and properties. These components are necessary to comprehend how OOP promotes effective and modular program design. Classes and objects are important to OOP. A class acts as a template for building objects, combining behavior and data into a single unit. Classes are represented by objects, which are examples of certain things with special characteristics and abilities. Because of this link, real-world circumstances may be modeled, which improves program intuitiveness and aligns it with our worldview.

In summary, the interplay between classes, objects, methods, fields, and properties forms the backbone of Object-Oriented Programming. These concepts not only streamline the development process but also enhance code organization, readability, and maintainability. As we continue to explore more advanced OOP techniques, mastering these basic concepts will be essential for building robust software solutions that can adapt to changing requirements and complexities in future projects.

Reference

Website

GeeksforGeeks. (2023, February 9). *Introduction of object oriented programming*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

OOP Terminology*: class, attribute, property, field, data member*. (n.d.). Stack Overflow. https://stackoverflow.com/questions/16751269/oop-terminology-class-attribute-property-field-data-member

Lesson 8: Object-Oriented Programming. (n.d.). https://www.oracle.com/java/technologies/oop.html

W3Schools.com. (n.d.). https://www.w3schools.com/java/java_oop.asp

What is object-oriented programming (OOP)? (n.d.). https://www.tutorialspoint.com/What-is-object-oriented-programming-OOP