In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

**“FOUR PRINCIPLES OF**

**OBJECT ORIENTED PROGRAMMING”**

Presented to:

Dr. Unife O. Cagas

Professor

Prepared by:

Galgo, Alyn Mae C.

BSCS 2A2 Student

**"Gadget Management System: Laptops, Smartphones, and Smartwatches - An Exploration in Python"**

Project Title

**PROJECT DESCRIPTION**

DESCRIPTION:

The provided code defines a set of classes for representing different types of gadgets. The main class is called `Gadget`, which serves as a blueprint for creating various types of gadgets. It has attributes for storing the name, brand, and price of a gadget, as well as a method for displaying information about the gadget.Three subclasses inherit from the `Gadget` class: `Smartphone`, `Laptop`, and `Smartwatch`. Each subclass represents a specific type of gadget and adds additional attributes and methods specific to that type of gadget.

The `Smartphone` class represents a smartphone and includes an additional attribute for storing the model of the smartphone. It overrides the `display\_info` method of the `Gadget` class to include information about the smartphone model.The `Laptop` class represents a laptop and includes an additional attribute for storing the laptop's screen size. Similar to the `Smartphone` class, it overrides the `display\_info` method of the `Gadget` class to include information about the laptop screen size. The `Smartwatch` class represents a smartwatch and includes an additional attribute for indicating whether the smartwatch is water-resistant. It also overrides the `display\_info` method of the `Gadget` class to include information about the water resistance.The code creates instances of each subclass (`Smartphone`, `Laptop`, and `Smartwatch`) and displays information about each gadget using the `display\_info` method.

This code demonstrates the use of inheritance and method overriding in object-oriented programming to model different types of gadgets with shared and specific characteristics.

**OBJECTIVES**

**1.** **Modular Code Structure**

The primary objective of the code is to implement a modular structure using classes and inheritance. Each class represents a specific type of gadget, allowing for easy organization and maintenance of code.

**2.** **Code Reusability**

By utilizing inheritance, the code aims to promote code reusability. The `Gadget` class serves as a base class, providing common attributes and methods that are shared among all gadget types. Subclasses inherit these attributes and methods, reducing code duplication and promoting efficient development.

**3.** **Customization for Specific Gadget Types:**

The code aims to provide customization for different types of gadgets. Each subclass (`Smartphone`, `Laptop`, `Smartwatch`) extends the functionality of the `Gadget` class by adding attributes specific to the respective gadget type, such as model, screen size, and water resistance.

**4.** **Information Display**

The code includes a method `display\_info` in each class to present information about each gadget type in a standardized format. This objective ensures consistency in displaying gadget information across different types.

**5.** **Encapsulation of Data and Behavior**

An objective of the code is to encapsulate data and behavior within each class. Each class encapsulates data attributes (such as name, brand, price) and behavior (such as displaying gadget information) within its scope, promoting code organization and maintainability.

**6. Demonstration of Object-Oriented Principles:**

The code tries to illustrate inheritance, encapsulation, polymorphism, and abstraction—four fundamental ideas of object-oriented programming (OOP). These ideas improve the readability, scalability, and maintainability of code.

**IMPORTANCE AND CONTRIBUTION OF THE PROJECT**

**1. Enhanced Code Organization**

The project contributes to improved code organization by implementing a structured approach to modeling different types of gadgets using classes and inheritance. This structured codebase makes it easier to understand, maintain, and extend the functionality in the future.

**2. Reusable Code Components**

The project promotes code reusability by utilizing inheritance and encapsulation. The base `Gadget` class provides common attributes and methods that can be inherited by subclasses, reducing redundant code and promoting efficient development.

**3. Scalability and Flexibility**

The project lays the foundation for scalability and flexibility in handling various types of gadgets. New types of gadgets can be easily added as subclasses, leveraging the existing structure and functionality provided by the base `Gadget` class.

**4. Standardized Information Display**

The project ensures consistency in displaying information about different types of gadgets through the `display\_info` method. This standardized approach simplifies the user interface and enhances the user experience by presenting gadget information in a uniform format.

**5. Educational Value**

The project serves as a valuable educational resource for learning and understanding key concepts of object-oriented programming (OOP), including inheritance, encapsulation, polymorphism, and abstraction. It provides practical examples of these concepts in action, helping developers improve their programming skills.

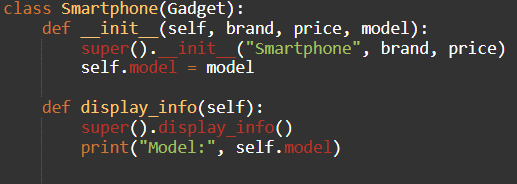
**6. Real-world Application**

The project demonstrates the application of OOP principles in real-world scenarios, such as modeling electronic gadgets. This real-world context enhances the relevance and practicality of the project, making it beneficial for developers seeking to apply OOP concepts in their own projects.

**FOUR PRINCIPLES OF OOP WITH CODE**

**ENHERITANCE**

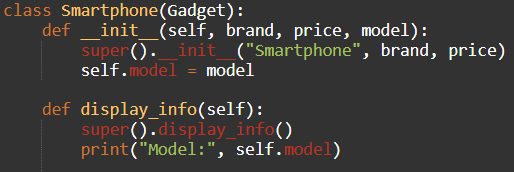
The `Smartphone` class extends the `Gadget` class, inheriting its attributes and methods, and further specializes the functionality to represent smartphones. In its constructor method, `\_\_init\_\_`, the class initializes the smartphone's attributes, including the brand, price, and a specific model. By calling the superclass's constructor using `super().\_\_init\_\_("Smartphone", brand, price)`, it ensures that the inherited attributes are properly initialized. Additionally, it initializes the `model` attribute specific to smartphones. The `display\_info` method of the `Smartphone` class provides a specialized way to display information about smartphones. It first calls the `display\_info` method of the superclass (`Gadget`) to print generic gadget information such as name, brand, and price. Then, it prints the specific model of the smartphone. This encapsulation of data and behavior within the `Smartphone` class allows for a clear representation of smartphones and promotes code organization and reuse, adhering to the principles of object-oriented programming.

****

**ENCAPSULATION**

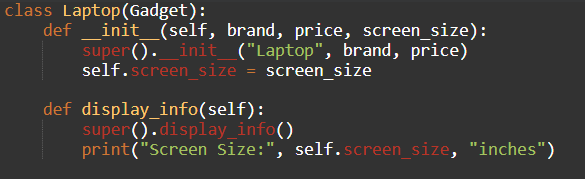
In the `Smartphone` class, encapsulation is exemplified through the structuring of data and behavior within the class. The constructor method, `\_\_init\_\_`, is utilized to initialize the attributes specific to a smartphone object, namely `brand`, `price`, and `model`. By encapsulating these attributes within the class, they become intrinsic properties of each smartphone instance, isolated from the rest of the program. Furthermore, the `\_\_init\_\_` method invokes the constructor of the superclass `Gadget`, utilizing inheritance while encapsulating the initialization process.

The method `display\_info` further illustrates encapsulation by encapsulating the behavior of displaying smartphone information. Within this method, information about the smartphone, including its model, is printed. The call to `super().display\_info()` within `display\_info` encapsulates the common display functionality inherited from the superclass `Gadget`, ensuring consistency in how gadget information is presented across different subclasses. Thus, encapsulation in the `Smartphone` class is achieved through the encapsulation of data within instance attributes and the encapsulation of behavior within methods, facilitating modularity and abstraction in the object-oriented design.

****

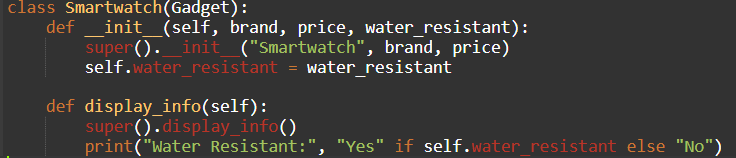
**POLYMORPHISM**

In the `Laptop` class, polymorphism is effectively demonstrated through the method overriding of `display\_info`. By inheriting from the `Gadget` superclass, the `Laptop` class shares a common interface with other gadget classes. However, despite having the same method signature as the `display\_info` method in the superclass, the `Laptop` class overrides this method with its own implementation tailored to laptops. When an instance of the `Laptop` class calls the `display\_info` method, it behaves differently compared to when an instance of the `Gadget` class calls the same method. This illustrates polymorphism in action, wherein the same method name exhibits different behaviors based on the specific subclass it is invoked from. This flexibility allows for the creation of specialized behavior in subclasses while maintaining a cohesive interface across related classes. Consequently.



**ABSTRACTION**

In the `Smartwatch` class, abstraction is evident through encapsulating implementation details and simplifying interaction with smartwatch objects. The `\_\_init\_\_` method initializes attributes like `brand`, `price`, and `water\_resistant`, abstracting their internal handling. By using `super().\_\_init\_\_("Smartwatch", brand, price)`, initialization of superclass attributes is abstracted.

Similarly, the `display\_info` method abstracts the behavior of presenting smartwatch details. Through `super().display\_info()`, common display functionality from the superclass is abstracted. Additionally, `print("Water Resistant:", "Yes" if self.water\_resistant else "No")` simplifies the representation of water resistance, abstracting complex data presentation. This abstraction enhances clarity and maintains a clean interface, promoting code readability and usability.

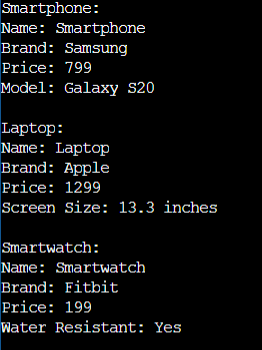
**SOFTWARE USED**

* [**https://www.onlinegdb.com/**](https://www.onlinegdb.com/)

**HARDWARE USED:**

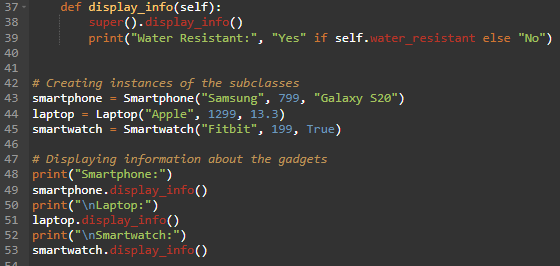
* MOBILE PHONE
* SCHOOL PC
* LAPTOP

**OUTPUT**

****

In the realm of technology, there are three noteworthy gadgets: a Samsung Galaxy S20 smartphone, an Apple laptop, and a Fitbit smartwatch. The Samsung Galaxy S20, priced at $799, is a top-tier smartphone boasting advanced features and exceptional performance under the reputable Samsung brand. Its sleek design and powerful capabilities position it as a coveted device in the market. Moving on to the laptop, we have an Apple offering priced at $1299. With a screen size of 13.3 inches, it combines portability with high performance, embodying Apple's trademark blend of elegance and functionality. Lastly, the Fitbit smartwatch, priced at $199, is a fitness-oriented wearable from the renowned brand. Its water-resistant design makes it suitable for use during workouts and outdoor activities, catering to health-conscious consumers seeking seamless integration of technology into their active lifestyles. Each of these devices represents the pinnacle of innovation in their respective categories, offering consumers a diverse array of options to meet their technological needs and preferences.

**CODE**

****

**USER GUIDE: GADGET MANAGEMENT SYSTEM**

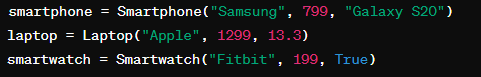
This code serves as a Gadget Management System, enabling the organization and display of information for laptops, smartphones, and smartwatches. Below is a detailed guide on how to utilize this system effectively:

**Understanding the Classes:**

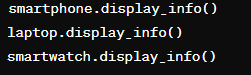
* Gadget: The base class encapsulates generic gadget attributes such as name, brand, and price. It includes a method display\_info() to print out these details.
* Smartphone, Laptop, Smartwatch: These subclasses inherit from Gadget. Each subclass adds unique attributes (model for Smartphone, screen\_size for Laptop, and water\_resistant for Smartwatch) and overrides the display\_info() method to include these additional details.

**Creating Instances:**

* Instantiate the subclasses (Smartphone, Laptop, or Smartwatch) to create instances of gadgets. Supply the required parameters for each gadget type.

EXAMPLE

**Displaying Information:**

* Utilize the display\_info() method to showcase information about each gadget instance.
* Usage:

**Interpreting Output:**

* The output provides a comprehensive overview of each gadget, detailing its name, brand, price, and any additional attributes specific to its subclass (e.g., model, screen size, water resistance).

**Customization:**

* Extend the system by introducing more subclasses or enhancing existing classes with additional functionalities to accommodate diverse gadget types and management requirements.

**Running the Code:**

* Copy the script into a Python environment and execute it to observe the displayed information for the created gadgets.

**REFERENCES**

1. <https://www.onlinegdb.com/>
2. <https://www.w3schools.com/>