Inheritance in OOP

What is Inheritance(is a relationship)

Inheritance is one of the fundamental concepts in object-oriented programming (OOP) that allows a new class (subclass or derived class) to inherit properties and behaviors (attributes and methods) from an existing class (base class or superclass). This allows you to create a new class that is a modified or specialized version of an existing class, promoting code reuse and establishing a hierarchical relationship between classes.

Inheritance is typically used to model an "is-a" relationship between classes, where the derived class is a more specific or specialized version of the base class. The derived class inherits the attributes and methods of the base class and can also have its own additional attributes and methods or override the inherited ones.

In [ ]:
```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass  # Base class method, to be overridden by subclasses

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"

# Creating objects
dog = Dog("Buddy")
cat = Cat("Whiskers")

# Calling the speak method on objects
print(dog.speak())
print(cat.speak())
```

```
Buddy says Woof!
Whiskers says Meow!
```

In this example:

- 'Animal' is the base class, and 'Dog' and 'Cat' are subclasses of 'Animal'.
- Both 'Dog' and 'Cat' inherit the 'name' attribute and the 'speak' method from the 'Animal' class.
- However, each subclass overrides the 'speak' method to provide its own implementation, representing the specific behavior of each animal. Inheritance is a powerful mechanism in OOP because it allows you to create a hierarchy of classes, with each level of the hierarchy adding or modifying functionality as needed. This promotes code reuse, encapsulation, and abstraction, making it easier to manage and extend your codebase.

Inheritance is a fundamental concept in object-oriented programming (OOP) that offers several benefits for software development. Here are five key advantages of inheritance:

1. **Code Reusability:** Inheritance allows you to reuse code from existing classes (base or parent classes) in new classes (derived or child classes). This promotes code reusability, reduces redundancy, and saves development time. Developers can leverage well-tested and established code when creating new classes.

2. **Hierarchical Structure:** Inheritance enables the creation of a hierarchical structure of classes, with each derived class inheriting attributes and methods from its parent class. This hierarchical organization makes it easier to understand and manage the relationships between different classes in a complex system.

3. **Promotes Polymorphism:** Inheritance is closely linked to the concept of polymorphism, which allows objects of different classes to be treated as objects of a common base class. This promotes flexibility and extensibility in your code, making it easier to work with diverse types of objects in a unified manner.

4. **Supports Code Extensibility:** With inheritance, you can extend existing classes to add or modify functionality. Derived classes can override inherited methods to provide specialized behavior while still benefiting from the base class's shared attributes and methods. This makes it straightforward to adapt and extend your code to accommodate changing requirements.

5. **Enhances Maintenance:** Inheritance improves code maintenance because changes made to the base class are automatically reflected in all its derived classes. This reduces the risk of introducing bugs during updates and ensures that modifications are consistently applied throughout the codebase. It also helps maintain a consistent interface for objects derived from the same base class.

In summary, inheritance is a powerful mechanism in OOP that promotes code reusability, structure, flexibility, extensibility, and ease of maintenance. It is a cornerstone of building complex software systems by organizing and leveraging existing code effectively.

Inheritance in summary

- A class can inherit from another class.

- Inheritance improves code reuse
- Constructor, attributes, methods get inherited to the child class
- The parent has no access to the child class
- Private properties of parent are not accessible directly in child class
- Child class can override the attributes or methods. This is called method overriding
- super() is an inbuilt function which is used to invoke the parent class methods and constructor

Types of Inheritance

1. **Single Inheritance:**
    - In single inheritance, a class inherits properties and behaviors from a single parent class. This is the simplest form of inheritance, where each class has only one immediate base class.
2. **Multilevel Inheritance:**
    - In multilevel inheritance, a class derives from a class, which in turn derives from another class. This creates a chain of inheritance where each class extends the previous one. It forms a hierarchy of classes.
3. **Hierarchical Inheritance:**
    - In hierarchical inheritance, multiple derived classes inherit from a single base or parent class. This results in a structure where several classes share a common ancestor.
4. **Multiple Inheritance (Diamond Problem):**
    - Multiple inheritance occurs when a class inherits properties and behaviors from more than one parent class. This can lead to a complication known as the "diamond problem," where ambiguity arises when a method or attribute is called that exists in multiple parent classes. Some programming languages, like Python, provide mechanisms to resolve this ambiguity.
5. **Hybrid Inheritance:**
    - Hybrid inheritance is a combination of two or more types of inheritance mentioned above. It is used to model complex relationships in a system where multiple inheritance hierarchies may intersect.

These different types of inheritance allow developers to model various relationships and structures in object-oriented programming. Choosing the appropriate type of inheritance depends on the specific requirements and design of the software being developed.

In [ ]:
```
# single inheritance
class Phone:
    def __init__(self, price, brand, camera):
```

```
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

SmartPhone(1000,"Apple","13px").buy()
```

```
Inside phone constructor
Buying a phone
```

In [ ]:
```
# multilevel
class Product:
    def review(self):
        print ("Product customer review")

class Phone(Product):
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass
```

In [ ]:
```
# Hierarchical
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass
```

In [ ]:
```
# Multiple
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class Product:
    def review(self):
        print ("Customer review")
```

```
class SmartPhone(Phone, Product):
    pass
```