Inheritance - MAD 102 Week 11 Notes

Hia Al Saleh

November 13th, 2024

Contents

1 Introduction to Inheritance

Inheritance is a fundamental concept in object-oriented programming that allows a class to inherit attributes and methods from another class, promoting code reuse and modularity. The class that inherits is known as the **derived class**, while the class it inherits from is the **base class**.

- **Benefits of Inheritance**:
  - Reduces redundancy by reusing code across multiple classes.
  - Allows for updates in the base class to automatically reflect in derived classes.
  - Simplifies the code structure by promoting a hierarchical organization.

2 Class Hierarchy

Inheritance creates a **class hierarchy**, which models relationships in the form of parent and child classes.

- **Parent (Base) Class**: Provides attributes and methods to be inherited by child classes.
- **Child (Derived) Class**: Inherits attributes and methods from the parent class and can introduce its own methods.

For example:

```python
class Animal:
    def speak(self):
        print("Some generic animal sound")

class Dog(Animal):  # Dog is a derived class of Animal
    def speak(self):
        print("Woof Woof")
```

In this example, `Dog` inherits from `Animal` and overrides the `speak` method.

3 Overriding Methods

A derived class can redefine or **override** a method inherited from the base class. This enables the derived class to provide a specialized behavior.

```python
class Animal:
    def speak(self):
        print("Some generic animal sound")

class Cat(Animal):
    def speak(self):
        print("Meow")

animal = Animal()
animal.speak()  # Outputs: Some generic animal sound

cat = Cat()
cat.speak()     # Outputs: Meow
```

Here, the `Cat` class overrides the `speak` method from `Animal` to provide a custom implementation.

4 Multiple Inheritance

In Python, a class can inherit from multiple base classes, which is called **multiple inheritance**. This is useful when a class needs features from more than one base class.

```
class Walker:
    def walk(self):
        print("Walking on two legs")

class Flyer:
    def fly(self):
        print("Flying with wings")

class Bird(Walker, Flyer):  # Bird inherits from both
    Walker and Flyer
    pass

sparrow = Bird()
sparrow.walk()  # Outputs: Walking on two legs
sparrow.fly()   # Outputs: Flying with wings
```

The `Bird` class inherits methods from both `Walker` and `Flyer`.

5 Mixins

Mixins are classes used to extend the functionality of a class by providing additional methods. They are not intended to be standalone and are usually used with multiple inheritance.

```
class StuntMixin:
    def jump(self, distance):
        print(f"Jumped {distance} meters!")

class CarefulMixin:
    def caution(self):
        print("Proceeding with caution.")

class DirtBike(StuntMixin, CarefulMixin):
    pass

dirt_bike = DirtBike()
dirt_bike.jump(10)       # Outputs: Jumped 10 meters!
dirt_bike.caution()       # Outputs: Proceeding with
    caution.
```

The `DirtBike` class uses `StuntMixin` and `CarefulMixin` to gain additional capabilities.

6 Extended Python Code Example

Below is an extended example of inheritance and mixins, combining all of the above concepts into a more complex class structure.

```
# Base class
class Vehicle:
    def __init__(self, current_speed=0):
        self.current_speed = current_speed

    def description(self):
        print(f"Traveling at {self.current_speed} km/h")

    def make_noise(self):
        print("Vroom Vroom")

# Derived class inheriting from Vehicle
class Bicycle(Vehicle):
    def __init__(self, has_basket=False):
        super().__init__(0)
        self.has_basket = has_basket

    def description(self):
        super().description()
        print("But in the bike lane!")

    def make_noise(self):
        print("Ring Ring")

# Mixins for additional functionality
class StuntMixin:
    def jump(self, distance):
        print(f"Jumped {distance} meters!")

    def skid(self, distance):
        print(f"Skidded {distance} meters!")
```

```python
class CarefulMixin:
    def avoid_obstacles(self):
        print("Carefully avoiding obstacles.")

# Multiple inheritance with Bicycle and mixins
class DirtBike(Bicycle, StuntMixin, CarefulMixin):
    def __init__(self, has_basket=False):
        super().__init__(has_basket)

    def make_noise(self):
        print("Braaaap")

# Demonstration of class functionality
car = Vehicle()
car.description()
car.make_noise()

bike = Bicycle(has_basket=True)
bike.description()
bike.make_noise()

dirt_bike = DirtBike()
dirt_bike.jump(5)
dirt_bike.skid(2)
dirt_bike.avoid_obstacles()
dirt_bike.current_speed = 20
dirt_bike.description()
dirt_bike.make_noise()
```

- **Vehicle**: The base class that provides a description and make_noise methods.
- **Bicycle**: A derived class that inherits from Vehicle, overriding the `make_noise` method and extending `description`.
- **StuntMixin** and **CarefulMixin**: Provide additional capabilities to the DirtBike class.
- **DirtBike**: Combines all features of Vehicle, Bicycle, and the mixins to create a versatile class with multiple behaviors.

7 Class Exercise

- Create a program that will store a list of contacts
- There are three types of contacts
- A person with a name and phone number
- A student with a name, a phone number, and the school they attend
- An employee with a name, a phone number, and the place they work
- A student is a person and an employee is also a person.

The program will ask what type of contact to enter, and will continually ask until the user says they are done. Then it will display all the contacts entered.

- If the contact is a person – it will display their name and phone number
- If the contact is a student – it will display their name and phone number and then the school they attend
- If the contact is an employee – it will display their work place followed by their name and phone number

7.1 Code

```
    # Person class
class Person:
    # Constructor
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone

    # String representation of the object
    def description(self):
        return f"{self.name} {self.phone}"

# Student class inherits from Person class
class Student(Person):
    # Constructor
    def __init__(self, name, phone, school):
        super().__init__(name, phone)
        self.school = school
    # String representation of the object
    def description(self):
        return f"{self.name} {self.phone} {self.school}"

# Employee class inherits from Person class
class Employee(Person):
```

```
          # Constructor
          def __init__(self, name, phone, work):
                      super().__init__(name, phone)
                      self.work = work
          # String representation of the object
          def description(self):
                    return f"{self.work} {self.name} {self
                        .phone}"
# Main program
contacts = []

while True:
          contact_type = input("Enter type of contact (
              person, student, employee) or done: ").lower
              ()
          if contact_type == "done":
                    break
          name = input("Enter name: ")
          phone = input("Enter phone number: ")
          if contact_type == "person":
                    contact = Person(name, phone)
          elif contact_type == "student":
                    school = input("Enter school: ")
                    contact = Student(name, phone, school)
          elif contact_type == "employee":
                    work = input("Enter work place: ")
                    contact = Employee(name, phone, work)
          contacts.append(contact)

for contact in contacts:
          print(contact)
```