```python
class MyClass:
    # Class variables (shared among all instances)
    class_variable = "I am a class variable"

    # Constructor method (initializer)
    def __init__(self, name):
        # Instance variable
        self.name = name

    # Instance method
    def say_hello(self):
        print(f"Hello, my name is {self.name}")

# Creating an instance of MyClass
obj1 = MyClass("Alice")

# Accessing instance variables and calling instance methods
print(obj1.name)        # Output: Alice
obj1.say_hello()        # Output: Hello, my name is Alice

# Accessing class variables
print(MyClass.class_variable)  # Output: I am a class variable
```

A Class is a blueprint for creating objects, defining their structure and behavior, while an object is an instance of a class, representing a specific realization of that blueprint with its own state and behavior. Classes provide a way to organize and encapsulate code, while objects represent concrete instances of that code that can be manipulated and interacted with.

Example: If you have a class called Car, it may have attributes like color, brand, and model, and methods like drive(), stop(), etc.

Example: If you create an object my_car from the Car class, it would represent a specific car with its own color, brand, model, etc.

super():

super() is used to initialize the parent class part of the instance, ensuring that the parent class is properly set up.

```python
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def make_sound(self):
        raise NotImplementedError("Subclasses must implement this method")
```

```
        # or
        pass

    def __str__(self):
        return f"{self.name} is a {self.species}"


class Dog(Animal):
    def __init__(self, name, breed):
        # Initialize the base class part of the instance
        super().__init__(name, "Dog")
        self.breed = breed

    def make_sound(self):
        return "Woof!"

    def __str__(self):
        return f"{self.name} is a {self.breed} {self.species}"

generic_animal = Animal("Generic", "Unknown")
dog = Dog("Buddy", "Golden Retriever")

# This will not raise an error, but it won't provide any useful feedback either
print(generic_animal.make_sound())  # Output: None
print(dog.make_sound())  # Output: Woof!
print(dog)
```

In the Dog class, the init method calls super().__init__(name, "Dog") to initialize the name and species attributes using the **init** method of the Animal class.

This ensures that the Animal part of the Dog instance is properly initialized without having to duplicate the initialization code.