OOP in Python (Part 3)

Abstraction

Abstraction is a concept in object-oriented programming that focuses on hiding the implementation details of a class from the user. It allows you to define the essential features and behaviors of an object while hiding the unnecessary complexity. Abstraction helps in simplifying code and promoting modularity.

Soyutlama, nesne yönelimli programlamadaki bir kavramdır ve bir sınıfın uygulama detaylarını kullanıcıdan gizleme üzerine odaklanır. Nesnenin temel özelliklerini ve davranışlarını tanımlamanıza olanak tanırken gereksiz karmaşıklığı gizler. Soyutlama, kodu basitleştirmeye ve modülerliği desteklemeye yardımcı olur.

In [ ]:

```python
# Abstract Base Class (ABC)
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

# Concrete class that implements the Shape abstract class
# Shape soyut sınıfını uygulayan somut sınıf (Concrete class)
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

# Concrete class that implements the Shape abstract class
# Shape soyut sınıfını uygulayan somut sınıf (Concrete class)
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

    def perimeter(self):
        return 2 * 3.14 * self.radius

# Function that works with any Shape object
# Herhangi bir Shape nesnesiyle çalışan bir fonksiyon
def print_shape_details(shape):
    print(f"Area: {shape.area()}")
    print(f"Perimeter: {shape.perimeter()}")

# Creating instances of the concrete classes
# Somut sınıfların örnekleri oluşturuluyor
rectangle = Rectangle(5, 4)
circle = Circle(4)

# Calling the function with different types of objects
# Farklı tiplerde nesnelerle fonksiyonun çağrılması
print("Rectangle Details:")
print_shape_details(rectangle)
print("\nCircle Details:")
print_shape_details(circle)
```

```
Rectangle Details:
Area: 20
Perimeter: 18

Circle Details:
Area: 50.24
Perimeter: 25.12
```

In this example, we have an abstract base class Shape that defines two abstract methods area and perimeter. Abstract methods are declared using the abstractmethod decorator, and they don't have any implementation in the abstract class.

We also have two concrete classes, Rectangle and Circle, which inherit from the Shape abstract class. These classes implement the area and perimeter methods with their own specific calculations for calculating the area and perimeter.

The print_shape_details function takes a Shape object as a parameter and calls the area and perimeter methods on it. Since Shape is an abstract class, we cannot create instances of it directly. However, we can use its concrete subclasses (Rectangle and Circle) to pass objects of those classes to the function.

This is abstraction in action – the implementation details of the Shape class are hidden from the user, and they can work with Shape objects without needing to know the specific implementations of area and perimeter for each subclass. Abstraction allows us to focus on the essential features of objects and helps in building more maintainable and scalable code.

Bu örnekte, soyut bir temel sınıf olan Shape'in alan (area) ve çevre (perimeter) adında iki soyut metodunu tanımladığı bir durum var. Soyut metodlar, abstractmethod dekoratörü kullanılarak bildirilir ve soyut sınıfta herhangi bir uygulamaya sahip değildir.

Ayrıca, Shape soyut sınıfından miras alan Rectangle ve Circle adında iki somut sınıfımız var. Bu sınıflar, alan ve çevre metodlarını kendi özel hesaplamalarıyla uygularlar.

print_shape_details fonksiyonu, bir Shape nesnesini parametre olarak alır ve onun üzerinde alan ve çevre metodlarını çağırır. Shape soyut bir sınıf olduğundan, doğrudan onun örneklerini oluşturamayız. Ancak, bu soyut sınıfın somut alt sınıflarını (Rectangle ve Circle) kullanarak bu sınıfların nesnelerini fonksiyona iletebiliriz.

Bu, soyutlama örneğidir - Shape sınıfının uygulama detayları kullanıcıdan gizlenir ve kullanıcılar, her alt sınıf için alan ve çevre özel uygulamalarını bilmeye ihtiyaç duymadan Shape nesneleriyle çalışabilirler. Soyutlama, nesnelerin temel özelliklerine odaklanmamızı sağlar ve daha bakımı kolay ve ölçeklenebilir kodlar oluşturmaya yardımcı olur.

Encapsulation

Encapsulation is a fundamental concept in object-oriented programming that involves bundling the data (attributes) and methods that operate on that data within a single unit, known as a class. This helps in protecting the data from direct access and modification by external code, ensuring that data integrity and consistency are maintained. Access to the data is controlled through methods, known as getters and setters, which provide an interface to interact with the class's attributes.

Onur_Gumus