اصول شیءگرایی

اصول شیءگرایی

در برنامه‌نویسی شیءگرا، به‌منظور طراحی سیستم‌های پیچیده و مقیاس‌پذیر، چهار اصل بنیادین وجود دارد که رعایت آن‌ها منجر به ایجاد کدهایی ساختارمند، خوانا و قابل نگه‌داری می‌شود. اگرچه ممکن است این اصول در فرآیند توسعه نرم‌افزار به‌طور ناخودآگاه به کار گرفته شوند، اما آگاهی از آن‌ها موجب درک عمیق‌تر و بهره‌گیری هدفمند از قابلیت‌های شیءگرایی خواهد شد. این اصول عبارتند از:

- **Abstraction (انتزاع)**
- **Encapsulation (کپسوله‌سازی)**
- **Inheritance (وراثت)**
- **Polymorphism (چندریختی)**

Abstraction

برای درک مفهوم انتزاع، می‌توان عملکرد تلویزیون را مثال زد. کاربر تنها از طریق دکمه‌های کنترل از راه دور، عملیات مورد نظر خود، نظیر تغییر کانال یا تنظیم صدا، را انجام می‌دهد، بی‌آنکه نیاز به دانستن مکانیسم‌های داخلی تلویزیون داشته باشد. به همین ترتیب، در دنیای نرم‌افزار، انتزاع موجب می‌شود که پیچیدگی‌های داخلی سیستم پنهان شده و تنها جنبه‌های ضروری در اختیار کاربران یا توسعه‌دهندگان قرار گیرد. بدین ترتیب، Abstraction بستری فراهم می‌کند که افراد بدون نیاز به درک کامل از جزئیات فنی، بتوانند با سیستم تعامل کنند.

Encapsulation

مفهوم کپسوله‌سازی به شیوه‌ای از طراحی اشاره دارد که در آن، داده‌ها و جزئیات پیاده‌سازی یک شیء از دید کاربران خارجی پنهان نگه داشته می‌شوند و تنها از طریق رابط‌های مشخص، امکان تعامل با آن فراهم می‌شود. به عنوان نمونه، دستگاه خودپرداز (ATM) اطلاعات حساس کاربران و منطق داخلی پردازش تراکنش‌ها را از دسترس عموم مخفی نگه می‌دارد و صرفاً از طریق مجموعه‌ای از ورودی‌های کنترل‌شده، همچون ورود کارت و رمز عبور، به کاربران امکان استفاده از خدمات را می‌دهد. کپسوله‌سازی علاوه بر افزایش امنیت، موجب کاهش

وابستگی بخش‌های مختلف سیستم به یکدیگر شده و در نهایت از بروز تغییرات ناخواسته یا سوءاستفاده‌های احتمالی جلوگیری می‌کند.

تفاوت Abstraction و Encapsulation

هر دو مفهوم انتزاع و کپسوله‌سازی به شیوه‌هایی برای پنهان‌سازی جزئیات سیستم اشاره دارند، اما تفاوت آن‌ها در نحوه به‌کارگیری این پنهان‌سازی است. در حالی که Abstraction بر ساده‌سازی تعامل کاربر با سیستم و ارائه یک رابط کاربری شفاف تأکید دارد، Encapsulation بیشتر به مدیریت سطح دسترسی و محدودسازی نحوه تغییر یا مشاهده داده‌های داخلی سیستم تمرکز می‌کند.

Inheritance

وراثت یکی از مفاهیم کلیدی در شیءگرایی است که امکان انتقال ویژگی‌ها و رفتارها از یک شیء (کلاس والد) به دیگر اشیاء (کلاس‌های فرزند) را فراهم می‌کند. این مفهوم را می‌توان به ارث‌بردن ویژگی‌های ژنتیکی و رفتاری در خانواده‌ها تشبیه کرد، جایی که صفاتی مانند رنگ چشم، مهارت‌های خاص یا عادات رفتاری از نسل‌های قبل به نسل‌های بعد منتقل می‌شوند. وراثت موجب تسهیل در توسعه نرم‌افزار، کاهش تکرار کد و ایجاد قابلیت گسترش‌پذیری در سیستم‌های بزرگ می‌شود.

Polymorphism

چندریختی، مفهومی است که به امکان ارائه‌ی یک عملکرد واحد به روش‌های مختلف اشاره دارد. به عنوان نمونه، فردی که به چندین زبان مسلط است، می‌تواند یک پیام واحد را به زبان‌های مختلف بیان کند، به‌گونه‌ای که هر شنونده بر اساس زبان خود آن را درک نماید. در برنامه‌نویسی، Polymorphism این قابلیت را فراهم می‌کند که یک تابع یا متد واحد بتواند بر روی انواع مختلف داده عمل کند. برای مثال، تابع `len` در زبان پایتون هم برای رشته‌ها و هم برای لیست‌ها به کار گرفته می‌شود؛ در هر دو حالت، هدف تابع شمارش تعداد عناصر است، اما نحوه‌ی پیاده‌سازی آن برای هر نوع داده متفاوت خواهد بود.

1. Abstraction (انتزاع)

تعریف: انتزاع یعنی پنهان کردن جزئیات پیچیده و نمایش تنها جنبه‌های ضروری به کاربران. این کار کمک می‌کند تا تعامل با سیستم ساده‌تر و کاربرپسندتر شود.

مثال: فرض کنید یک برنامه برای کنترل تلویزیون می‌نویسید. کاربر تنها با استفاده از دکمه‌های کنترل از راه دور، می‌تواند تلویزیون را روشن کند یا کانال را تغییر دهد. او نیازی به دانستن نحوه عملکرد داخلی تلویزیون ندارد. در اینجا، ما جزئیات پیچیده‌تر مانند نحوه پردازش سیگنال‌های تلویزیونی را از کاربر پنهان می‌کنیم.

```
class Television:
    def __init__(self):
        self._is_on = False

    def power(self):
        self._is_on = not self._is_on
        state = "ON" if self._is_on else "OFF"
        print(f"Television is {state}")

# Using abstraction
tv = Television()
tv.power()  # The TV turns on or off without needing to know complex deta
```

2. Encapsulation (کپسوله‌سازی)

تعریف: کپسوله‌سازی به این معنی است که داده‌ها و روش‌های داخلی شیء از دیدگاه بیرونی پنهان می‌شوند و تنها از طریق متدهای خاص به آن‌ها دسترسی پیدا می‌شود.

مثال: تصور کنید که شما یک دستگاه خودپرداز (ATM) دارید. برای استفاده از این دستگاه، کاربر باید کارت و رمز عبور را وارد کند. جزئیات عملیات درونی (مثل انتقال پول از حساب‌ها) برای کاربر پنهان است.

```
class ATM:
    def __init__(self, balance):
        self._balance = balance  # The data is hidden
        self._pin = "1234"  # The password is hidden

    def authenticate(self, pin):
        if pin == self._pin:
            print("Authentication successful")
            return True
        else:
```

```
            print("Invalid PIN")
            return False

    def withdraw(self, amount, pin):
        if self.authenticate(pin):
            if amount <= self._balance:
                self._balance -= amount
                print(f"Withdrew {amount}. New balance: {self._balance}")
            else:
                print("Insufficient funds")
        else:
            print("Failed to authenticate")

# Using encapsulation
atm = ATM(1000)
atm.withdraw(200, "1234")  # The user cannot access the internal data
```

3. Inheritance (وراثت)

تعریف: وراثت به این معنی است که ویژگی‌ها و رفتارهای یک کلاس (کلاس والد) به کلاس‌های دیگر (کلاس فرزند) منتقل می‌شود. این امر کدنویسی را ساده‌تر کرده و قابلیت گسترش‌پذیری را افزایش می‌دهد.

مثال: در یک سیستم مدرسه، کلاس `Person` می‌تواند ویژگی‌هایی مانند نام و سن را داشته باشد. کلاس‌های `Student` و `Teacher` می‌توانند از کلاس `Person` ویژگی‌ها و رفتارهای مشترک را به ارث ببرند.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        return f"Hello, my name is {self.name}."

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

```

```
    def study(self):
        print(f"{self.name} is studying.")

class Teacher(Person):
    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def teach(self):
        print(f"{self.name} is teaching {self.subject}.")

# Using inheritance
student = Student("Alice", 20, "S123")
teacher = Teacher("Bob", 40, "Math")
print(student.greet())  #Inherited attribute
student.study()  #Specific behavior of the Student class
teacher.teach()  # Specific behavior of the Teacher class
```

4. Polymorphism (چندریختی)

تعریف: چندریختی به این معنی است که یک تابع یا متد می‌تواند بر روی انواع مختلف داده‌ها عمل کند و نتیجه متفاوتی بدهد. این ویژگی به برنامه‌نویس این امکان را می‌دهد که از یک متد برای انواع مختلف شیء استفاده کند.

مثال: تابع `area()` می‌تواند مساحت یک دایره یا یک مستطیل را محاسبه کند. با اینکه اسم متد یکی است، اما خروجی متفاوتی دارد.

```
class Shape:
    def area(self):
        pass  # Base method, does not perform any action

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
```

```
        return 3.14 * self.radius * self.radius

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

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

# Using polymorphism
shapes = [Circle(5), Rectangle(4, 6)]
for shape in shapes:
    print(f"Area: {shape.area()}")  # The same area method, but for diffe
```