Setter و Getter

کنترل سطح دسترسی در برنامه‌نویسی شیءگرا

در برنامه‌نویسی شیءگرا، داده‌های درون کلاس باید به گونه‌ای مدیریت شوند که امنیت و انسجام آن‌ها حفظ شود. برای این کار، سه سطح دسترسی متداول در زبان‌های برنامه‌نویسی وجود دارد:

۱. **عمومی (Public)**: متغیرها و متدهایی که در این سطح تعریف می‌شوند، از هرجای برنامه قابل دسترسی هستند. این سطح معمولاً برای متدهایی استفاده می‌شود که قرار است به‌عنوان واسط کاربری کلاس عمل کنند.
۲. **محافظت‌شده (Protected)**: این متغیرها و متدها که با یک آندرلاین (_) مشخص می‌شوند، نشان می‌دهند که نباید مستقیماً از بیرون کلاس استفاده شوند، اما همچنان در کلاس‌های فرزند قابل‌دسترسی‌اند. این روش برای زمانی کاربرد دارد که بخواهیم برخی ویژگی‌ها را در ارث‌بری حفظ کنیم اما همچنان تغییر مستقیم از خارج کلاس را محدود کنیم.
۳. **خصوصی (Private)**: متغیرها و متدهایی که با دو آندرلاین (__) شروع می‌شوند، فقط از داخل همان کلاس قابل‌دسترسی هستند و دسترسی مستقیم از خارج کلاس امکان‌پذیر نیست. این سطح از دسترسی معمولاً برای داده‌هایی به کار می‌رود که نیاز به حفاظت بیشتری دارند.

مثال سطح دسترسی در پایتون:

```python
class MyClass:
    public_var = 5       # Public
    _protected_var = 10  # Protected
    __private_var = 15   # Private
```

در این مثال، `public_var` در دسترس همه است، `_protected_var` در کلاس‌های فرزند قابل استفاده است، و `__private_var` فقط از داخل کلاس قابل‌دسترسی است.

چرا به Getter و Setter نیاز داریم؟

در بسیاری از موارد، نمی‌خواهیم متغیرهای خصوصی مستقیماً تغییر کنند. مثلاً اگر بخواهیم مقدار یک متغیر را محدود کنیم (مانند سن یک فرد یا موجودی یک حساب)، نیاز داریم که تغییر آن از طریق یک روش کنترل‌شده انجام شود. اینجاست که **Getter** و **Setter** به کمک ما می‌آیند.

پیاده‌سازی Getter و Setter به روش استاندارد

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

    def get_age(self):
        return self.__age

    def set_age(self, age):
        if 0 <= age <= 120:
            self.__age = age
        else:
            print("Invalid!")
```

استفاده:

```
p = Person(25)
print(p.get_age())  # 25
p.set_age(130)  # Invalid!
```

استفاده از @property برای Getter و Setter

پایتون راهی مختصرتر برای تعریف Getter و Setter دارد:

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

    @property
    def age(self):
```

```
        return self._age

    @age.setter
    def age(self, value):
        if 0 <= value <= 120:
            self._age = value
        else:
            print("Invalid")
```

استفاده:

```
p = Person(30)
print(p.age)  # 30
p.age = 150   # Invalid
```

کاربرد عملی Getter و Setter

بیایید یک مثال کاربردی‌تر ببینیم. فرض کنید یک کلاس **بانک** داریم که موجودی حساب را مدیریت می‌کند و نباید مقدار آن منفی شود:

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance

    @property
    def balance(self):
        return self._balance

    @balance.setter
    def balance(self, amount):
        if amount >= 0:
            self._balance = amount
        else:
            print("Balance cannot be negative!")
```

استفاده:

```
account = BankAccount(500)
print(account.balance)  # 500
account.balance = -100  # Balance cannot be negative!
```

نتیجه‌گیری

دو تابع Getter و Setter به ما کمک می‌کنند کنترل بیشتری روی متغیرهای خصوصی داشته باشیم.

استفاده از `@property` کد را خواناتر و حرفه‌ای‌تر می‌کند.

با استفاده از Getter و Setter می‌توانیم قوانین خاصی برای تغییر مقادیر متغیرها اعمال کنیم.

این روش‌ها باعث بهبود امنیت، خوانایی و پایداری کدهای ما در برنامه‌نویسی شیءگرا می‌شوند!