Robert C. Martin SOLID prensipleri kaç tane ve nelerdir ?

**SOLID**, yazılım tasarımında kullanılan ve Robert C. Martin (Uncle Bob) tarafından tanıtılan **beş temel prensibi** ifade eder. Bu prensipler, yazılımın daha esnek, bakımı kolay, anlaşılır ve genişletilebilir olmasını sağlar. SOLID, aşağıdaki beş prensibin baş harflerinden oluşur:

**1. S - Single Responsibility Principle (Tek Sorumluluk Prensibi)**

* **Tanım**: Bir sınıf veya fonksiyon, yalnızca **bir sorumluluğa** sahip olmalıdır.
* **Amaç**: Kodun daha anlaşılır, bakımı kolay ve yeniden kullanılabilir olmasını sağlamak.
* **Örnek**:

python

Copy

# Yanlış: Hem rapor oluşturma hem de raporu kaydetme

class Report:

def generate\_report(self, data):

# Rapor oluşturma

pass

def save\_report(self, report):

# Raporu kaydetme

pass

# Doğru: Her sınıf tek bir sorumluluğa sahip

class ReportGenerator:

def generate\_report(self, data):

# Rapor oluşturma

pass

class ReportSaver:

def save\_report(self, report):

# Raporu kaydetme

pass

**2. O - Open/Closed Principle (Açık/Kapalı Prensibi)**

* **Tanım**: Bir sınıf veya modül, **genişletmeye açık** ancak **değişikliğe kapalı** olmalıdır.
* **Amaç**: Mevcut kodu değiştirmeden yeni özellikler ekleyebilmek.
* **Örnek**:

python

Copy

# Yanlış: Her yeni şekil eklenmesi mevcut kodu değiştirir

class AreaCalculator:

def calculate\_area(self, shape):

if shape == "rectangle":

return width \* height

elif shape == "circle":

return 3.14 \* radius \* radius

# Doğru: Yeni şekiller eklemek için mevcut kodu değiştirmeye gerek yok

class Shape:

def area(self):

pass

class Rectangle(Shape):

def area(self):

return self.width \* self.height

class Circle(Shape):

def area(self):

return 3.14 \* self.radius \* self.radius

**3. L - Liskov Substitution Principle (Liskov'un Yerine Geçme Prensibi)**

* **Tanım**: Bir alt sınıf (subclass), üst sınıfın (superclass) yerine geçebilmeli ve bu değişiklik programın doğruluğunu bozmamalıdır.
* **Amaç**: Kalıtımın doğru kullanılmasını sağlamak.
* **Örnek**:

python

Copy

# Yanlış: Kuş sınıfı, Penguen için uygun değil

class Bird:

def fly(self):

pass

class Penguin(Bird):

def fly(self):

raise Exception("Penguins can't fly!")

# Doğru: Uçabilen kuşlar ve uçamayan kuşlar ayrı sınıflar

class Bird:

pass

class FlyingBird(Bird):

def fly(self):

pass

class Penguin(Bird):

def swim(self):

pass

**4. I - Interface Segregation Principle (Arayüz Ayrıştırma Prensibi)**

* **Tanım**: Bir sınıf, kullanmadığı arayüzleri (interface) uygulamaya zorlanmamalıdır. Arayüzler, küçük ve özelleştirilmiş olmalıdır.
* **Amaç**: Gereksiz bağımlılıkları önlemek ve kodun daha esnek olmasını sağlamak.
* **Örnek**:

python

Copy

# Yanlış: Tek bir büyük arayüz

class Machine:

def print(self):

pass

def scan(self):

pass

def fax(self):

pass

# Doğru: Küçük ve özelleştirilmiş arayüzler

class Printer:

def print(self):

pass

class Scanner:

def scan(self):

pass

class FaxMachine:

def fax(self):

pass

**5. D - Dependency Inversion Principle (Bağımlılıkların Tersine Çevrilmesi Prensibi)**

* **Tanım**: Üst seviye modüller, alt seviye modüllere bağlı olmamalıdır. Her ikisi de **soyutlamalara** (abstractions) bağlı olmalıdır.
* **Amaç**: Bağımlılıkları azaltmak ve sistemin daha esnek olmasını sağlamak.
* **Örnek**:

python

Copy

# Yanlış: Üst seviye modül, alt seviye modüle bağlı

class LightBulb:

def turn\_on(self):

pass

def turn\_off(self):

pass

class Switch:

def \_\_init\_\_(self, bulb):

self.bulb = bulb

def operate(self):

self.bulb.turn\_on()

# Doğru: Her ikisi de soyutlamaya bağlı

class SwitchableDevice:

def turn\_on(self):

pass

def turn\_off(self):

pass

class LightBulb(SwitchableDevice):

def turn\_on(self):

pass

def turn\_off(self):

pass

class Switch:

def \_\_init\_\_(self, device):

self.device = device

def operate(self):

self.device.turn\_on()

**SOLID Prensiplerinin Faydaları:**

1. **Esneklik**: Kod, yeni gereksinimlere kolayca uyum sağlar.
2. **Bakım Kolaylığı**: Kod daha anlaşılır ve düzenlidir.
3. **Yeniden Kullanılabilirlik**: Modüller bağımsız olduğu için farklı projelerde kullanılabilir.
4. **Test Edilebilirlik**: Her bir modül bağımsız olarak test edilebilir.

**Özet:**

SOLID prensipleri, yazılım tasarımında **kaliteli ve sürdürülebilir** kod yazmak için temel bir rehberdir. Bu prensipleri uyguladığınızda, yazılımınız daha **modüler**, **esnek** ve **bakımı kolay** hale gelir.