**SINAVA HAZIRLIK NOTU**  
**Konu: SOLID Prensipleri**

**📄 GİRİŞ**

SOLID, yazılım geliştirirken **daha bakım yapılabilir**, **esnek** ve **anlaşılabilir** sistemler kurmak için kullanılan 5 temel prensibi ifade eder.

**S.O.L.I.D:**

1. **S**ingle Responsibility Principle (SRP)
2. **O**pen/Closed Principle (OCP)
3. **L**iskov Substitution Principle (LSP)
4. **I**nterface Segregation Principle (ISP)
5. **D**ependency Inversion Principle (DIP)

**✅ 1. Single Responsibility Principle (SRP)**

Bir sınıfın yalnızca **tek bir sorumluluğu** olmalıdır.

**📄 Amaç:**

* Her sınıf **yalnızca bir iş yapsın** ve onu iyi yapsın.

**🔴 Kötü Örnek:**

class UserManager:

def process\_user(self, name, email):

self.name = name

self.email = email

print(f"Saving {self.name} to database")

print(f"Sending email to {self.email}")

* Bu sınıf hem veri kaydı, hem e-posta gönderimi yapıyor → Çok sorumluluk!

**🔵 İyi Örnek:**

class User:

def \_\_init\_\_(self, name, email):

self.name = name

self.email = email

class UserRepository:

def save\_user(self, user):

print(f"Saving {user.name} to database")

* Her sınıf sadece **tek işe odaklanıyor.**

**✅ 2. Open/Closed Principle (OCP)**

Sınıflar **genişletmeye açık**, ancak **değiştirmeye kapalı** olmalıdır.

**📄 Amaç:**

* Yeni davranış eklemek için mevcut kodu değiştirmeden **uzatma** (inheritance ya da interface) kullan.

**🔴 Kötü Örnek:**

class PaymentProcessor:

def process\_payment(self, type, amount):

if type == "paypal": ...

elif type == "credit": ...

* Yeni ödeme tipi eklemek için sınıf değiştirilmek zorunda.

**🔵 İyi Örnek:**

class PaymentMethod:

def process(self, amount):

pass

class PayPal(PaymentMethod):

def process(self, amount): print("PayPal")

class Processor:

def pay(self, method: PaymentMethod, amount):

method.process(amount)

* Yeni ödeme tipi **sadece yeni sınıfla** eklenir.

**✅ 3. Liskov Substitution Principle (LSP)**

Tüm alt sınıflar, **ana sınıf gibi davranabilmelidir.**

**📄 Amaç:**

* Alt sınıf kullanılırsa sistemin davranışı bozulmamalı.

**🔴 Kötü Örnek:**

class GiftCardPayment(Payment):

def process(self, amount):

if amount > 50:

raise ValueError("Limit aşıldı")

* Ana sınıf her miktarı işliyor gibi ama bu alt sınıf hata fırlatıyor → **bozulma!**

**🔵 İyi Örnek:**

class GiftCardPayment(Payment):

def process(self, amount):

amount = min(amount, 50)

return f"Processed {amount}"

* Alt sınıf, ana sınıfın beklentisini bozmaz.

**✅ 4. Interface Segregation Principle (ISP)**

Sınıflar, **kullanmadığı metotlara sahip arayüzleri implement etmek zorunda kalmamalıdır.**

**📄 Amaç:**

* “**Şişman interface**” yerine **birden çok küçük interface** kullan.

**🔴 Kötü Örnek:**

interface IVehicle {

void Drive();

void Fly();

}

class Car : IVehicle {

void Fly() { throw NotImplementedException(); }

}

* Araba uçamaz ama zorla Fly() var.

**🔵 İyi Örnek:**

interface ICar { void Drive(); }

interface IAirplane { void Fly(); }

class Car : ICar { void Drive() { ... } }

* Herkes sadece ihtiyacını alır.

**✅ 5. Dependency Inversion Principle (DIP)**

Üst seviye sınıflar, alt seviye sınıflara **bağlı olmamalı**. Her ikisi de bir **soyutlamaya bağlı** olmalı.

**📄 Amaç:**

* Kodun test edilebilirliğini artırmak
* Detayları değiştirirken ana mantığı bozmamak

**🔴 Kötü Örnek:**

class PetLover:

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

self.sound\_maker = DogBarker() # Dog'a bağlı!

**🔵 İyi Örnek:**

class PetLover:

def \_\_init\_\_(self, pet\_sound: PetSound):

self.sound\_maker = pet\_sound

Artık hem kedi, hem tavşan verebiliriz.

**✨ Genel Tavsiye:**

SOLID prensipleri ile:

* Kodun **modüler yapısı** artar,
* **Bakım kolaylaşır**,
* **Test yazmak kolaylaşır**,
* Ekip içinde iş bölümü daha net olur.