Nesne Yönelimli Tasarım İlkeleri Nedir?

Çalışma Kağıdı

Nesne yönelimli tasarım ilkeleri, Robert C. Martin tarafından popülerleştirilen SOLID prensiplerini kapsar: Tek Sorumluluk (SRP), Açıklık/Kapalılık (OCP), Yerine Koyma (LSP), Arayüz Ayrımı (ISP) ve Bağımlılığın Tersine Çevrilmesi (DIP). Bu ilkeler, sürdürülebilir ve ölçeklenebilir yazılım sistemleri oluşturmak için rehberlik eder.

Sorular

1. Hangi SOLID prensibi, bir sınıfın yalnızca bir ana sorumluluğa sahip olması gerektiğini belirtir?

A) Açıklık/Kapalılık Prensibi
B) Tek Sorumluluk Prensibi
C) Yerine Koyma Prensibi
D) Arayüz Ayrımı Prensibi

2. Yazılım varlıklarının genişlemeye açık ancak değiştirmeye kapalı olması gerektiğini belirten prensip hangisidir?

A) Liskov Yerine Koyma Prensibi
B) Bağımlılığın Tersine Çevrilmesi Prensibi
C) Açıklık/Kapalılık Prensibi
D) Arayüz Ayrımı Prensibi

3. Hangi prensip, türetilmiş sınıfların temel sınıflarının yerine sorunsuz bir şekilde geçebilmesini gerektiğini söyler?

A) Tek Sorumluluk Prensibi
B) Arayüz Ayrımı Prensibi
C) Bağımlılığın Tersine Çevrilmesi Prensibi
D) Liskov Yerine Koyma Prensibi

4. Tek Sorumluluk Prensibi (SRP) bir sınıfa neden uygulanmalıdır?

5. Açıklık/Kapalılık Prensibi (OCP) yazılımın esnekliğini nasıl artırır?

6. Bağımlılığın Tersine Çevrilmesi Prensibi (DIP) soyutlamaları nasıl kullanır?

7. Tanımla: SOLID prensiplerinin açılımı nedir?

8. Tanımla: Tek Sorumluluk Prensibi (SRP) neyi amaçlar?

9. Tanımla: Açıklık/Kapalılık Prensibi (OCP) neyi vurgular?

Cevap Anahtarı

1. B) Tek Sorumluluk Prensibi — Tek Sorumluluk Prensibi (SRP), bir sınıfın yalnızca bir ana sorumluluğa sahip olması gerektiğini savunur.

2. C) Açıklık/Kapalılık Prensibi — Açıklık/Kapalılık Prensibi (OCP), yazılım varlıklarının genişlemeye açık ancak değiştirmeye kapalı olması gerektiğini belirtir.

3. D) Liskov Yerine Koyma Prensibi — Liskov Yerine Koyma Prensibi (LSP), türetilmiş sınıfların temel sınıflarının yerine sorunsuz bir şekilde geçebilmesi gerektiğini ifade eder.

4. Bir sınıfın yalnızca bir ana sorumluluğu olmalıdır. Bu, sınıfın yalnızca bir nedenle değişmesi gerektiği anlamına gelir. Eğer bir sınıf birden fazla sorumluluğa sahipse, bu sorumlulukları farklı sınıflara ayırmak, kodun daha anlaşılır, test edilebilir ve bakımı daha kolay olmasını sağlar.

5. Yazılım varlıkları (sınıflar, modüller, fonksiyonlar vb.) genişlemeye açık, ancak değiştirmeye kapalı olmalıdır. Bu, mevcut kodu değiştirmeden yeni işlevsellik eklemeyi mümkün kılarak hataları azaltır ve geliştirme sürecini hızlandırır.

6. Üst düzey modüller alt düzey modüllere bağlı olmamalıdır; her ikisi de soyutlamalara bağlı olmalıdır. Soyutlamalar ise detaylara bağlı olmamalıdır; detaylar soyutlamalara bağlı olmalıdır. Bu, sistemin daha gevşek bağlı olmasını ve bağımlılıkların yönetilmesini kolaylaştırır.

7. Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.

8. Bir sınıfın yalnızca bir görevi olmasını sağlamak.

9. Mevcut kodu değiştirmeden yeni özellikler ekleyebilme yeteneği.