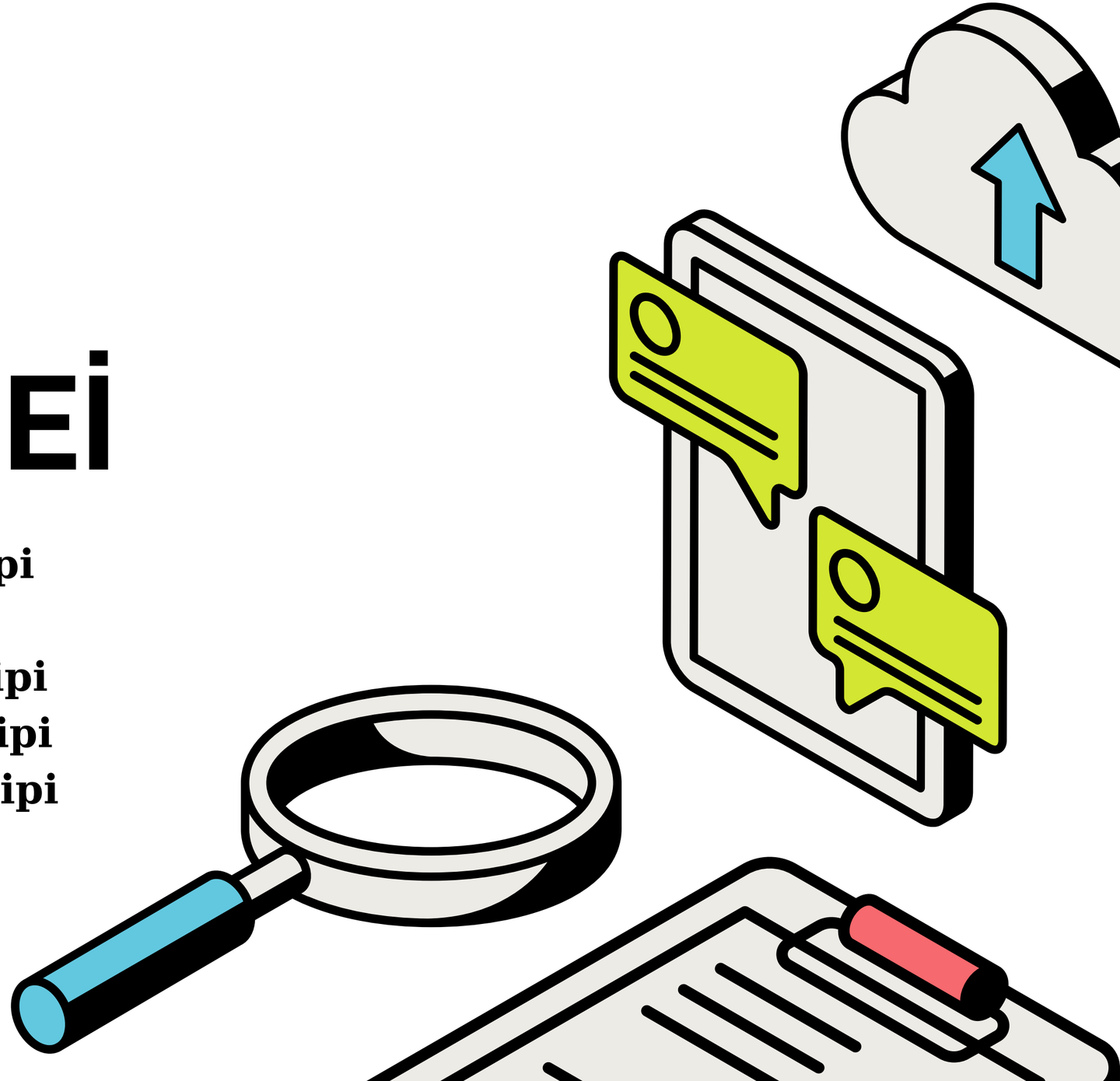
SOLİD PRENSİPLEREİ

- **(S)ingle Responsibility Prensipi**
- **(O)pen/Closed Principle**
- **(L)iskov ‘s Substitution Prensipi**
- **(I)nterface Segregation Prensipi**
- **(D)ependency Inversion Prensipi**

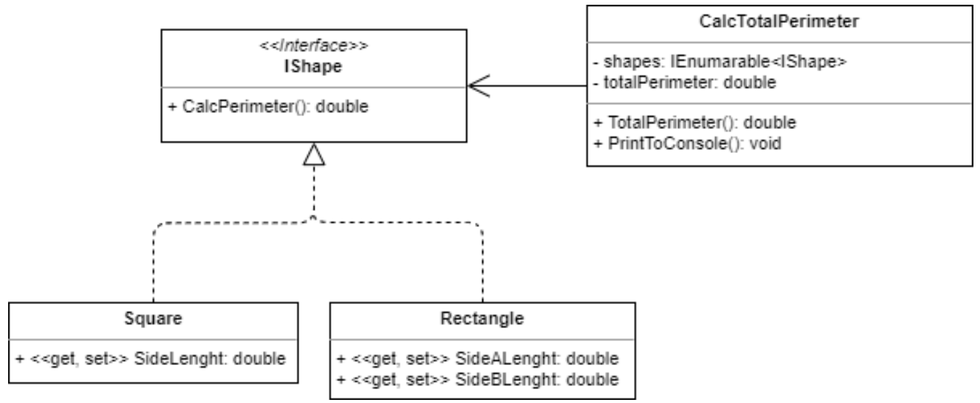


Tarih:16/07//2024

BÜŞRA KARA

Single Responsibility Prensibi (SRP)

Single Responsibility Principle (SRP) kısaca her bir sınıf veya fonksiyonun yalnızca bir işlevi yerine getirmesi gerektiğini belirtir. Bu prensip, yazılım geliştirme süreçlerinde kodun daha temiz, daha okunabilir ve daha yönetilebilir olmasını sağlamayı amaçlar.



- Sınıflar için: Bir sınıf, bir tek sorumluluğa (göreve) odaklanmalıdır. Bu, sınıfın sadece bir tür işi yapması gerektiği anlamına gelir. Örneğin, bir Müşteri sınıfı yalnızca müşteri bilgilerini tutmalı ve bu bilgileri yönetmelidir. Müşteri sınıfı aynı zamanda fatura oluşturma veya veri tabanı işlemleri yapmamalıdır.

- Fonksiyonlar için: Bir fonksiyon, yalnızca bir işlevi gerçekleştirmelidir. Fonksiyonun amacı, aldığı parametreleri işleyip bir sonuç üretmektir. Tek Sorumluluk İlkesi'ne göre, bir fonksiyon birden fazla işi yapmamalı veya farklı durumlar için farklı işlevler yerine getirmemelidir.

Bu ilke, kodun daha okunabilir, bakımı daha kolay ve hata ayıklaması daha basit olmasını sağlar. Ayrıca, kod parçalarının yeniden kullanılabilirliğini artırır ve bir değişiklik gerektiğinde sadece ilgili birimlerin etkileneceği anlamına gelir, diğer bağımlılıkları minimumda tutarak yan etkileri azaltır.

Özetle, SRP, yazılım geliştirme süreçlerinde modülerlik ve sürdürülebilirlik sağlamak için temel bir prensiptir.

(O)pen/Closed Prensibi

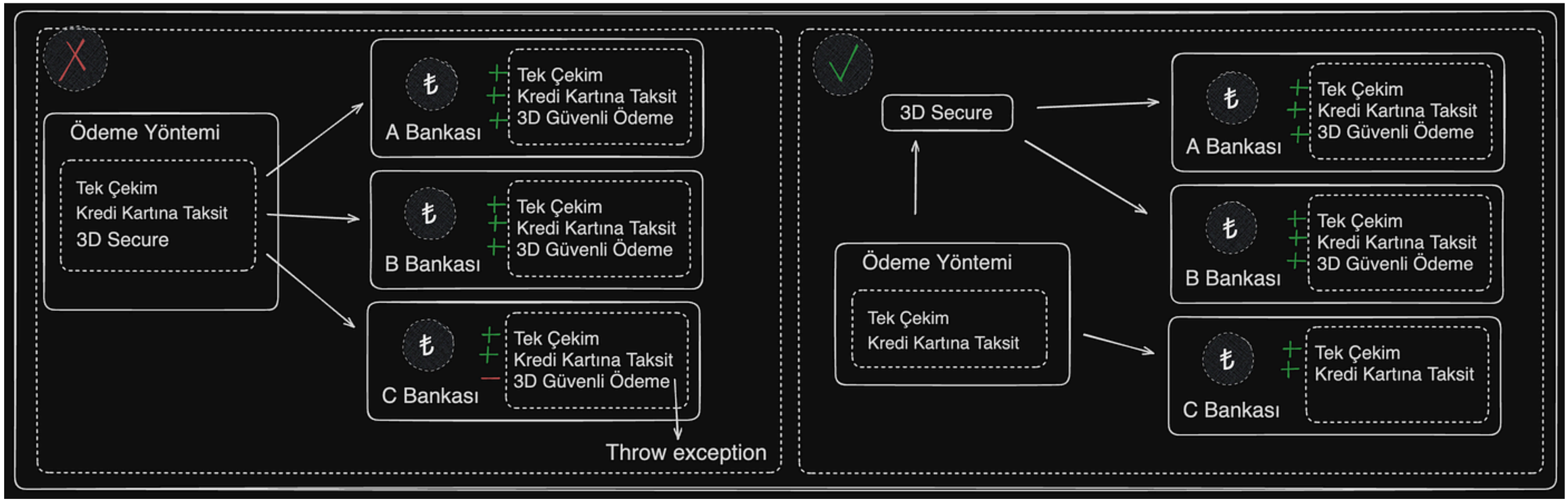
Açık-Kapalı Prensibi'nin uygulanması, yazılımın modüler olmasını ve birimler arasındaki bağımlılıkların minimize edilmesini sağlar. Bu da kodun daha bakımı kolay, yeniden kullanılabilir ve test edilebilir olmasını sağlar. Yazılım geliştiriciler için uzun vadede zaman ve kaynak tasarrufu sağlar, ayrıca yazılımın esnekliğini ve adaptasyon yeteneğini artırır.

Açık Olma (Open for Extension): Yazılım, yeni işlevselliğin eklenmesine ve var olanın genişletilmesine izin vermelidir. Değişen iş gereksinimlerine veya yeni kullanıcı ihtiyaçlarına yanıt verebilmek için yazılımın kolayca genişletilebilmesi önemlidir. Örneğin, bir e-ticaret platformunda yeni bir ödeme yöntemi eklemek veya yeni bir ürün kategorisi tanıtmak, yazılımın genişletilebilir olması sayesinde sorunsuz bir şekilde yapılabilir.

Kapalı Olma (Closed for Modification): Mevcut kod, var olan işlevselliği değiştirmeyecek şekilde tasarlanmalıdır. Yani, mevcut kod parçalarını veya modülleri değiştirmeden, yeni işlevselliği eklemek mümkün olmalıdır. Bu, kodun istikrarını ve güvenilirliğini artırır çünkü mevcut işleyiş bozulmadan yeni özellikler eklenir. Örneğin, bir ödeme işlemleri kütüphanesinde, mevcut ödeme yöntemleri işleyişini değiştirmeden yeni bir ödeme işlemi entegrasyonu yapılabilir.

Liskov ‘s Substitution Prensibi

Liskov'un Yerine Geçme Prensibi (Liskov Substitution Principle - LSP), yazılım tasarımında kalıtım (inheritance) ilişkilerinin doğru ve tutarlı bir şekilde kullanılmasını vurgular. Bu prensip, bir alt sınıftan oluşturulan nesnelerin, üst sınıfın nesneleriyle yer değiştirdiklerinde aynı davranışı göstermek zorunda olduğunu belirtir. Yani, alt sınıf nesneleri, üst sınıfın tüm işlevselliğini korumalı ve beklenen şekilde çalışmalıdır.



Interface Segregation Prensibi

- **Kullanıcı Odaklı Tasarım**: İSP, bir arayüzün, kullanıcıları için anlam ifade eden ve ihtiyaçlarına uygun olan yöntemleri içermesi gerektiğini vurgular. Kullanıcılar, kullandıkları arayüzlerin sunduğu yöntemleri anlamalı ve kullanmalıdır.
- **Ayrıştırma (Segregation)**: Arayüzlerin genel amaçlı olmaktan ziyade belirli işlevleri veya işlev gruplarını hedeflemesi gerektiğini belirtir. Böylece, bir arayüzün mümkün olan en spesifik ve az sayıda yöntemi içermesi teşvik edilir.
- **Sınıfların Yalıtılması**: Arayüzlerin sınıflar arasındaki ilişkilerde ve bağımlılıklarda yalıtılmasını sağlar. Böylece, bir sınıf, sadece ihtiyaç duyduğu belirli arayüzleri uygular ve bu sayede gereksiz bağımlılıklardan kaçınabilir.

Dependency Inversion Prensibi

Yüksek seviye sınıflarda bir davranış değiştiğinde, alt seviye davranışların bu değişime uyum sağlaması gerekir. Ancak, düşük seviye sınıflarda bir davranış değiştiğinde, üst seviye sınıfların davranışında bir bozulma meydana gelmemelidir.

Kaynakça

- http://cagataykiziltan.net/solid-prensipleri/
- https://omereryilmaz.com/single-responsibility-principle/
- https://chatgpt.com/
- https://tr.linkedin.com/pulse/react-a%C3%A7%C4%B1k-kapal%C4%B1-prensibi-ocp-digital-vizyon-akademi#:~:text=A%C3%A7%C4%B1k%2DKapal%C4%B1%20Prensibi%20(OCP)%2C%20yaz%C4%B1l%C4%B1m%C4%B1n%20dinamik%20bir%20%C5%9Fekilde,i%C5%9Flevlere%20kapal%C4%B1%20olmas%C4%B1%20gerekti%C4%9Fini%20belirtir.
- http://www.kurumsaljava.com/2009/10/16/open-closed-principle-ocp-acik-kapali-tasarim-prensibi/
- https://medium.com/@gamzendemir/solid-prensi%CC%87pleri%CC%87-41c546acb8bd
- https://gokhana.medium.com/interface-segregation-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-ac0fd6812ecf
- https://gokhana.medium.com/dependency-inversion-prensibi-nedir-kod-%C3%B6rne%C4%9Fiyle-soli%CC%87d-b61296523565
- https://innovationm.co/solid-dependency-inversion-principle/