Bölüm 3

Kapsülleme

3.1 Giriş

Kapsülleme (encapsulation), nesne tabanlı programlamanın (OOP) temel prensiplerinden biridir ve bir nesnenin iç verilerini dış dünyadan gizleyerek yalnızca tanımlı arayüzler üzerinden erişime izin verir. Bu bölümde, kapsüllemenin tanımı, avantajları, erişim kontrolü mekanizmaları ve modern yazılım mühendisliğindeki rolü ele alınacaktır. Amaç, okuyucuların kapsüllemenin neden önemli olduğunu ve nasıl uygulandığını anlamasıdır.

3.2 Kapsülleme Nedir?

Kapsülleme, bir nesnenin veri ve davranışlarını bir araya getirerek verilere doğrudan erişimi kısıtlar ve yalnızca belirli metodlar (örneğin, getter ve setter metodları) aracılığıyla kontrollü erişim sağlar. Bu, veri gizleme (data hiding) ve erişim kontrolü (access control) ile gerçekleştirilir.

Örneğin, bir banka hesabı nesnesi düşünün: Hesap bakiyesi özel (private) bir özelliktir ve yalnızca para yatırma veya çekme metodları üzerinden değiştirilebilir.

Kapsüllemenin temel bileşenleri:

- **Özel Veri (Private Data)**:
  Nesnenin iç verileri, dış dünyadan gizlenir. Bu, nesnenin iç durumunun doğrudan değiştirilmesini önleyerek veri tutarlılığını korur. Örneğin, bir sınıfın içindeki değişkenler (fields) özel olarak tanımlanır, böylece yalnızca sınıfın kendi metodları tarafından erişilebilir hale gelir. Bu yaklaşım, dışarıdan gelebilecek hatalı müdahaleleri engeller ve nesnenin davranışını daha öngörülebilir kılar.
- **Erişim Belirleyiciler**:
  Public, private ve protected gibi anahtarlar, erişim seviyesini tanımlar.

  Public belirleyici, herkesin erişimine açıktır ve genellikle arayüz metodları için kullanılır. UML diyagramlarında + sembolü ile gösterilir. Sözde-kod ile **genel** ile ifade edilir.

Private belirleyici, yalnızca sınıfın kendisi tarafından erişilebilir, iç verileri korumak için idealdir. UML diyagramlarında - sembolü ile gösgerilir. Sözde-kod ile **özel** ile ifade edilir.

Protected belirleyici, sınıf ve alt sınıflar (miras alma durumunda) tarafından erişilebilir, kalıtım mekanizmalarında faydalıdır. UML diyagramlarında # sembolü ile gösterilir. Sözde-kod ile **korumalı** ile ifade edilir.

Bazı dillerde (örneğin, C#) **internal** gibi ek belirleyiciler de bulunur, ancak temel olarak bu üçü kapsüllemenin temelini oluşturur.

- **Getter/Setter Metodları**:
  Verilere kontrollü erişim sağlar. Getter metodları (örneğin, getBakiye()), veriyi okumak için kullanılır ve genellikle değeri doğrudan döndürür. Setter metodları (örneğin, setBakiye(miktar)) ise, veriyi güncellemek için kullanılır ve giriş değerini doğrular (örneğin, negatif değer kabul etmez). Bu metodlar, veri erişimini merkezi bir noktadan yöneterek kodun bakımını kolaylaştırır ve bazı dillerde otomatik property'ler (auto-properties) ile daha basit hale getirilebilir.

3.3 Kapsüllemenin Avantajları

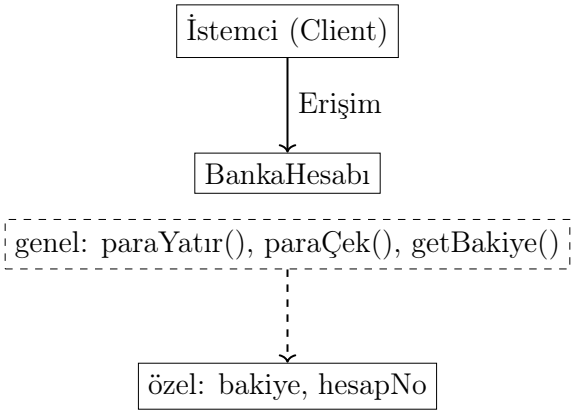
Kapsülleme, yazılım tasarımında şu avantajları sağlar:

- **Güvenlik**: Özel veriler, yetkisiz erişimden korunur. Bu, hassas bilgilerin (örneğin, kullanıcı şifreleri veya finansal veriler) dışarıdan doğrudan değiştirilmesini engeller. Örneğin, bir kullanıcı sınıfında şifre alanı private olarak tanımlanırsa, dışarıdan doğrudan erişim mümkün olmaz ve yalnızca şifre değiştirme metodu üzerinden güvenli bir şekilde güncellenebilir, böylece şifre karmaşıklık kurallarını uygulama fırsatı doğar.
- **Modülerlik**: Nesnenin iç yapısı değişse bile dış arayüz sabit kalır. Bu, yazılımı daha esnek hale getirir; iç implementasyon (uygulama) değiştirilebilirken, diğer modüller etkilenmez. Örneğin, bir veritabanı erişim sınıfında bağlantı detayları kapsüllenirse, veritabanı sağlayıcısı değiştiğinde (örneğin, MySQL'den PostgreSQL'e geçiş) yalnızca sınıfın içi güncellenir, ancak dış arayüz aynı kalır.
- **Bakım Kolaylığı**: Kod değişiklikleri yalnızca nesne içinde yapılır, diğer bileşenler etkilenmez. Bu, büyük ölçekli projelerde zaman ve çaba tasarrufu sağlar. Örneğin, bir grafik arayüzü sınıfında renk ayarları kapsüllenirse, renk temsilini RGB'den HSL'ye değiştirmek sadece sınıf içinde yapılır ve arayüzü kullanan kodlar bundan etkilenmez.
- **Hata Azaltma**: Kontrollü erişim, veri tutarlılığını korur (örneğin, negatif bakiye önleme). Bu, potansiyel hataları önler ve kodun güvenilirliğini artırır. Örneğin, bir stok yönetim sınıfında stok miktarı private tutulur ve yalnızca ekleme/çıkarma metodları üzerinden değiştirilirse, stokun negatif olması engellenebilir, böylece mantıksal hatalar minimize edilir.

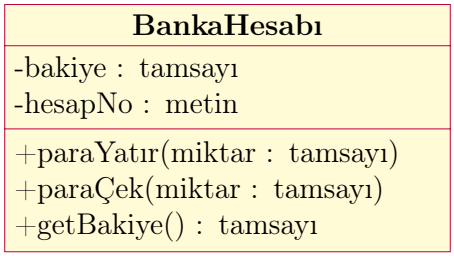
3.4 Erişim Kontrolü ve Uygulama

Kapsülleme, erişim belirleyiciler ve getter/setter metodları ile uygulanır.

Örneğin, bir *BankaHesabı* sınıfında bakiye özel (private) tutulur ve yalnızca public metodlar üzerinden erişilir. *BankaHesabı* için kapsülleme uygulayan bir UML diyagramı ve sözde-kod örneği verilmiştir:



Şekil 3.1 – Kapsülleme: İstemci ve Sınıf Arayüzü



Şekil 3.2 – BankaHesabı Sınıfının Kapsülleme ile UML Diyagramı

Sözde-kod 3.1 – Kapsülleme ile BankaHesabı Sınıfı

```
sınıf BankaHesabı {
 özel özellik bakiye: tamsayı
 özel özellik hesapNo: metin

 yapıcı(hesapNo: metin, başlangıçBakiye: tamsayı) {
  bu.hesapNo = hesapNo
  bu.bakiye = başlangıçBakiye
 }

 genel metod paraYatır(miktar: tamsayı) {
  eğer miktar > 0 ise
     bakiye = bakiye + miktar
  değilse
     hata "Geçersiz miktar"
 }

 genel metod paraÇek(miktar: tamsayı) {
  eğer miktar > 0 ve bakiye >= miktar ise
     bakiye = bakiye - miktar
  değilse
     hata "Yetersiz bakiye veya geçersiz miktar"
 }

 genel metod getBakiye(): tamsayı {
  döndür bakiye
 }
}

ana_program {
 hesap = yeni BankaHesabı("12345", 1000)
 hesap.paraYatır(500)
 hesap.paraÇek(200)
 yaz "Bakiye: ", hesap.getBakiye()  // Çıktı: Bakiye: 1300
}
```

3.5 Modern Bağlamda Kapsülleme

Kapsülleme, modern yazılım mühendisliğinde kritik bir rol oynar. Örneğin:

- Mikro Hizmetler: Her mikro hizmet, verilerini kapsülleyerek yalnızca tanımlı API'ler üzerinden iletişim kurar.
- API Tasarımı: REST veya GraphQL API'lerinde, veri erişimi kapsüllü endpoint'ler ile kontrol edilir.
- Güvenlik: Hassas veriler (örneğin, kullanıcı parolaları) özel tutularak yetkisiz erişim önlenir.

3.6 Alıştırmalar

1. Kapsüllemenin tanımını ve avantajlarını kendi kelimelerinizle açıklayın.
2. Bir *Kişi* sınıfı tasarlayın. Ad ve yaş özelliklerini özel tutarak, bu özelliklere erişim için getter/setter metodları yazın (sözde-kod ile).
3. Yukarıdaki *BankaHesabı* sınıfını temel alarak, bir *KrediKartı* sınıfı için sözde-kod yazın. Kredi limiti gibi bir özellik ekleyin ve kapsülleme uygulayın.
4. Şekil 3.2'deki UML diyagramını inceleyin ve başka bir sınıf (örneğin, *Araba*) için kapsüllemeyi gösteren bir UML diyagramı çizin.
5. Kapsüllemenin mikro hizmetlerde nasıl kullanıldığını araştırın ve bir örnekle açıklayın.