Bu raporda, **BillPaymentSystem** adlı yazılım uygulamasının dört temel nesne yönelimli programlama (OOP) prensibini nasıl kullandığını inceleyeceğiz. Bu prensipler şunlardır: **Kalıtım (Inheritance)**, **Kapsülleme (Encapsulation)**, **Soyutlama (Abstraction)** ve **Polimorfizm (Polymorphism)**.

**1. Kalıtım (Inheritance)**

**Kalıtım**, bir sınıfın (class) başka bir sınıftan özellikler ve davranışlar (metodlar) devralmasını sağlar. Bu yazılımda, **Bill** adında soyut bir sınıf, her bir fatura türü (Elektrik, Su, İnternet faturası) için bir temel sınıf olarak kullanılmıştır. Bu soyut sınıf, CalculateTotalAmount, GetBillId, GetCustomerName, GetDueDate ve ilgili setter metodlarını tanımlar.

Örneğin:

* **ElectricityBill**, **WaterBill** ve **InternetBill** sınıfları, **Bill** sınıfından türetilmiştir (inheritance). Bu sınıflar, temel **Bill** sınıfından devraldıkları özellikleri ve metodları kullanarak kendi özel işlevlerini (örneğin; elektrik tüketimi, su tüketimi veya internet abonelik ücreti hesaplama) gerçekleştirirler.

Kalıtım sayesinde, ortak özellikler ve metodlar bir kez yazılır ve sonra türetilen sınıflar tarafından kullanılır, bu da kodun tekrarlanmasını engeller.

**2. Kapsülleme (Encapsulation)**

**Kapsülleme**, sınıfların içindeki verileri (özellikler) doğrudan erişime kapalı hale getirir ve bu verilere sadece özel metodlar (getter ve setter) aracılığıyla erişim sağlanmasını sağlar. Bu şekilde, sınıfların dışarıdan erişilen verileri güvence altına alınmış olur.

Bu uygulamada:

* **Bill** sınıfı ve türevleri, **BillId**, **CustomerName** ve **DueDate** gibi özellikleri kapsüller. Bu özelliklere dışarıdan doğrudan erişilemez.
* Bu verilere yalnızca **getter** (örneğin GetBillId(), GetCustomerName()) ve **setter** (örneğin SetBillId(), SetCustomerName()) metodları aracılığıyla erişim sağlanır.

Örneğin, **ElectricityBill** sınıfı içinde **BillId** özelliği özel (private) olarak tanımlanmış ve bu değere yalnızca SetBillId() metodu ile set edilebilir. Bu, dışarıdan yapılan yanlış erişimlerin önüne geçer ve verilerin güvenliğini sağlar.

**3. Soyutlama (Abstraction)**

**Soyutlama**, gereksiz detayları gizler ve yalnızca önemli bilgileri kullanıcıya sunar. Bu, kullanıcının karmaşık sistemin iç yapısına müdahale etmeden basit bir arayüzle işlemler yapmasını sağlar.

Bu uygulama:

* **Bill** soyut sınıfı, tüm fatura türleri için temel bir yapı sağlar. CalculateTotalAmount(), GetBillId(), GetCustomerName(), ve GetDueDate() gibi soyut metodlar, türetilen sınıflar tarafından uygulanacak şekilde tanımlanmıştır.
* Her bir faturaya ait detaylar (elektrik birim maliyeti, su tüketimi, internet abonelik ücreti) bu soyut sınıfın metotlarıyla soyutlanır. Bu, kullanıcıya her bir fatura türünü aynı şekilde işleme olanağı sunar, fakat her fatura türünün nasıl çalıştığını bilmesine gerek yoktur.

Örneğin:

* **ElectricityBill** ve **WaterBill** sınıflarındaki CalculateTotalAmount metodları, elektrik ve su faturaları için spesifik hesaplama detaylarını gizler. Kullanıcı, her bir faturanın hesaplanması gerektiğini bilse de, hesaplama metodunun nasıl çalıştığını bilmesine gerek yoktur.

**4. Polimorfizm (Polymorphism)**

**Polimorfizm**, bir sınıfın birden fazla farklı şekilde davranabilmesini sağlar. Bu, aynı metodun farklı nesnelerle farklı işlevler gerçekleştirmesini mümkün kılar.

Bu uygulama:

* **PrintBillDetails** metodu, farklı fatura türleri için aynı işlemi yapabilir. **ElectricityBill**, **WaterBill** ve **InternetBill** gibi farklı faturalar, PrintBillDetails metodunu çağırabilir ve farklı hesaplamalar ile sonuçlar döndürebilirler. Bu metod, her fatura türüne uygun olarak CalculateTotalAmount metodunu çağırır, ancak hangi türdeki faturanın hesaplanacağına göre farklı sonuçlar elde edilir.
* Örneğin, **ElectricityBill** nesnesi, elektrik faturası için birim maliyet ve tüketim bilgilerini kullanarak bir toplam hesaplama yaparken, **WaterBill** nesnesi su birim maliyetini ve tüketimi kullanarak farklı bir hesaplama yapar.

Polimorfizm, kullanıcıya her bir fatura türünü tek bir yöntemle işleme imkanı sunar, ancak her bir fatura türü kendi özel hesaplama işlevini yerine getirir.

**Sonuç**

Bu yazılımda, OOP prensiplerinin hepsi etkili bir şekilde kullanılmıştır:

1. **Kalıtım**, ortak özelliklerin ve metodların tek bir sınıfta toplanmasını sağlar ve bu sınıfların türetilerek spesifik işlevselliklerin eklenmesini mümkün kılar.
2. **Kapsülleme**, verilerin dışarıdan erişime kapalı olmasını sağlayarak güvenliği artırır.
3. **Soyutlama**, karmaşık detayları gizler ve kullanıcıya basit bir arayüz sunar.
4. **Polimorfizm**, aynı metodun farklı nesnelerle farklı davranmasını sağlar ve kodun esnekliğini artırır.