NESNEYE YÖNELİK PROGRAMLAMA

Ders İşleyişi
ve
Tarihsel ve mantıksal bağlamda nesneye yönelik programlama

Emir Öztürk

DERS İÇERİKLERİ - İLETİŞİM

- emirozturk@trakya.edu.tr
- personel.trakya.edu.tr/emirozturk
- https://github.com/emirozturk

DEĞERLENDİRME

- Genel Ortalama
  - Vize: %30
  - Proje: %20
  - Final: %50

KAYNAKLAR

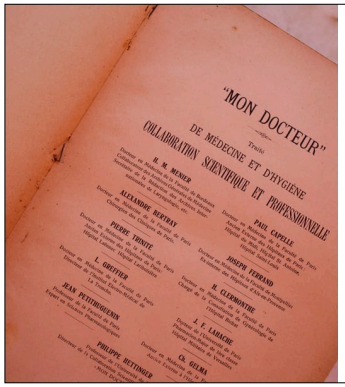
- Java How to Program Ninth Edition, Harvey M. Deitel & Paul J. Deitel, Prentice-Hall.
- UML Distilled, Martin Fowler, Addison-Wesley.

4

DERSİN AMACI

- Temel kavramlar
  - Programlama Dilleri
  - Nesneye Yönelik Programlama
    - Soyutlama
    - Kapsülleme
    - Çok biçimlilik
    - Kalıtım
- + Java

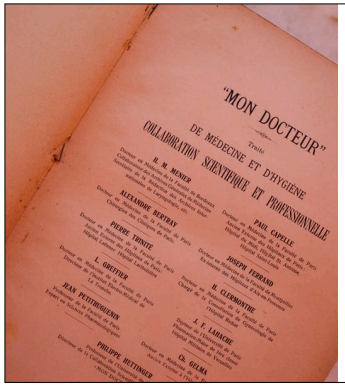
5



DERSİN KONULARI

- Programlama Paradigmaları
- Java Dilinin Genel Tanıtımı ve Syntax
- Sınıflar ve nesneler
- Yapıcılar, Statik, Final
- Referans Tabanlı ve Primitif Türler Soyutlama ve Sarmalama
- Overloading ve Overriding
- Arasınav

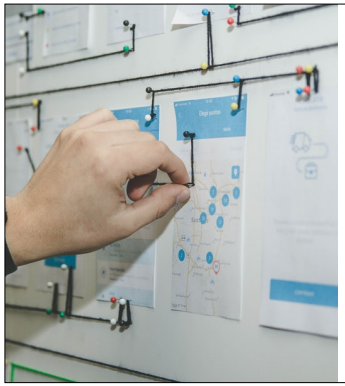
6



DERSİN KONULARI (DEVAM)

- Kalıtım
- UML ile Analiz ve Tasarım
- Soyutlama
- Soyut Sınıflar, Arayüzler ve Paketler
- Lambda İfadeleri
- Design Patterns
- Java ve Cpp

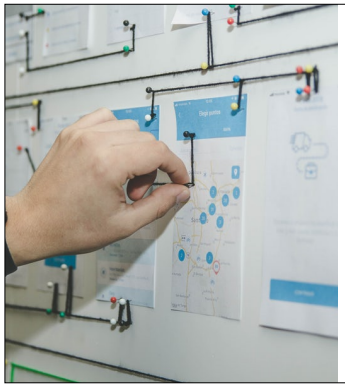
7



PROJENİN UYGULANMASI

- Kişi sayısı
  - Numara - Ad Soyad (1 Hafta)
- Proje Seçimi (1 Hafta)
  - Proje Listesi
  - Kendi Projeniz
  - Bir proje üç grup
- OTOMASYON**
- UNITY*****

8

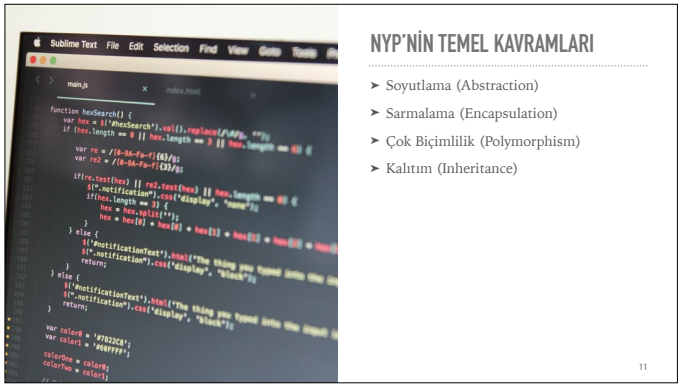


PROJENİN DEĞERLENDIRILMESI

- ⚠⚠⚠NESNEYE YÖNELİK⚠⚠⚠
- Kod teslimi
- Sunum
- Rapor

9

Programlama dili, bilgisayara belirli kurallar çerçevesinde komutlar vermeyi sağlayan formel bir yapıdır. Her programlama dili belirli sözdizimi (syntax) ve anlam (semantik) kurallarına sahiptir.
Programlama yaklaşımları zaman içinde farklı paradigmalar geliştirmiştir:
Imperatif Programlama Programın adım adım nasıl çalışacağını tanımlar. Kontrol akışı ve durum değişimi ön plandadır.
Prosedürel Programlama İşlemler fonksiyonlar veya prosedürler halinde organize edilir. Kod yeniden kullanılabilirliği artar.
Deklaratif Programlama Nasıl yapılacağından ziyade ne yapılması gerektiğini belirtir. SQL gibi diller bu yaklaşımı temsil eder.
Fonksiyonel Programlama Yan etkileri azaltmayı hedefleyen, fonksiyon kavramını merkeze alan yaklaşımdır.
Nesneye Yönelik Programlama (NYP) Gerçek dünyadaki varlıklar yazılımda nesneler olarak modellenir. Veri ve davranış birlikte ele alınır.

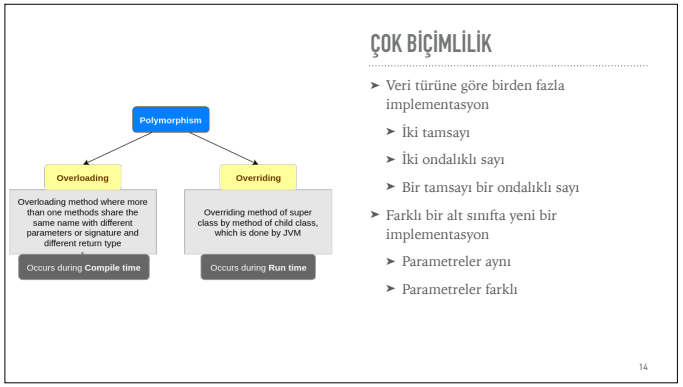


NYP dört temel kavram üzerine kuruludur:
1. Soyutlama (Abstraction) Gereksiz detayların gizlenerek yalnızca önemli özelliklerin ortaya çıkarılmasıdır.
2. Sarmalama (Encapsulation) Veri ve bu veriyi yöneten metotların birlikte tutulması ve dış dünyadan korunmasıdır.
3. Çok Biçimlilik (Polymorphism) Aynı isimdeki işlemlerin farklı durumlarda farklı davranışlar göstermesidir.
4. Kalıtım (Inheritance) Bir sınıfın özelliklerinin başka bir sınıf tarafından devralınmasıdır.

Bu kavramlar birlikte kullanıldığında daha sürdürülebilir ve modüler yazılımlar oluşturulur.

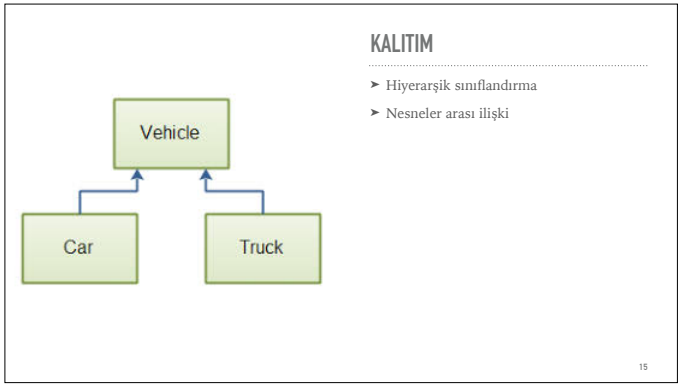
Soyutlama, nesneye yönelik programlamanın en temel kavramlarından biridir ve karmaşık sistemlerin daha anlaşılır hale getirilmesini amaçlar. Bir sistemi tasarlarken tüm detayları aynı anda görmek yerine, yalnızca önemli özelliklerin öne çıkarılması ve gereksiz ayrıntıların gizlenmesi prensibine dayanır. Böylece hem geliştirici hem de kullanıcı açısından sistemin yönetilmesi kolaylaşır. Yazılım geliştirme sürecinde soyutlama sayesinde bir nesnenin nasıl çalıştığı değil, ne yaptığı ön plana çıkarılır. Örneğin bir araba kullanıcısı motorun iç yapısını bilmeden aracı sürebilir. Yazılımda da benzer şekilde kullanıcı yalnızca gerekli arayüzü görür. Soyutlama veri bazlı veya işleyiş bazlı olabilir. Veri bazlı soyutlamada yalnızca gerekli veriler dışarıya açılırken, işleyiş bazlı soyutlamada metodun içindeki algoritma gizlenerek sadece sonuç odaklı kullanım sağlanır. Bu yaklaşım karmaşıklığın azaltılması, okunabilirliğin artırılması ve büyük sistemlerin daha modüler şekilde geliştirilmesi açısından kritik öneme sahiptir.

Sarmalama, verilerin ve bu veriler üzerinde işlem yapan fonksiyonların aynı yapı içinde bir araya getirilmesi ve dış dünyadan kontrolsüz erişime karşı korunması prensibidir. Nesneye yönelik programlamada güvenli ve sürdürülebilir kod yazmanın temel yollarından biri sarmalamadır. Bir sınıf içerisindeki değişkenlerin doğrudan erişime açılması yerine kontrollü erişim yöntemleri kullanılması, veri bütünlüğünü korur ve hatalı kullanımları engeller. Bu nedenle çoğu durumda veriler private olarak tanımlanır ve erişim getter veya setter metotları aracılığıyla sağlanır. Sarmalama sayesinde bir sınıfın iç yapısı dışarıdan bağımsız hale gelir; böylece sınıfın iç implementasyonu değişse bile dışarıdan kullanım şekli değişmeyebilir. Bu özellik büyük projelerde bakım maliyetini azaltır ve yazılımın daha güvenilir hale gelmesini sağlar. Ayrıca modülerlik ve tekrar kullanılabilirlik gibi yazılım mühendisliği prensiplerini destekleyen önemli bir mekanizmadır.

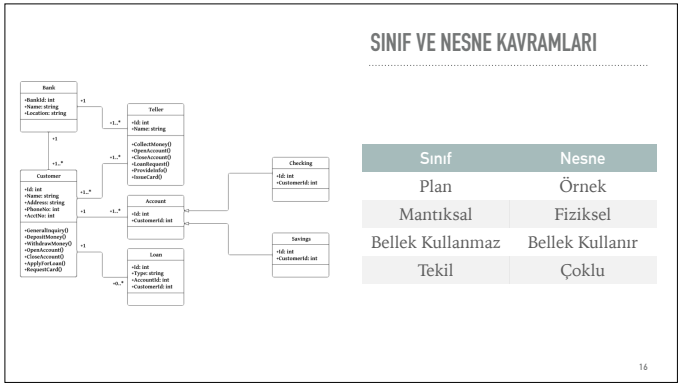


Çok biçimlilik, aynı isimdeki bir işlemin farklı durumlarda farklı davranışlar gösterebilmesini sağlayan nesneye yönelik programlama özelliğidir. Bu kavram, yazılımın esnekliğini artırarak farklı veri türleri veya farklı nesne tipleri için ortak bir arayüz kullanılmasına imkan verir. Örneğin aynı isimde bir metot farklı parametre türleriyle çağrıldığında farklı işlemler gerçekleştirebilir veya bir üst sınıfta tanımlanan bir davranış alt sınıflarda yeniden tanımlanarak farklı biçimde uygulanabilir. Overloading olarak bilinen yaklaşımda aynı isimli metotlar farklı parametre yapılarıyla aynı sınıf içinde tanımlanırken, overriding yaklaşımında alt sınıf üst sınıftan miras aldığı davranışı kendi ihtiyacına göre değiştirir. Çok

biçimlilik sayesinde geliştirilen sistemler daha genişletilebilir hale gelir, yeni sınıflar eklendiğinde mevcut kodun büyük ölçüde değiştirilmesi gerekmez. Bu durum özellikle büyük ölçekli yazılımlarda bakım ve geliştirme süreçlerini önemli ölçüde kolaylaştırır.



Kalıtım, mevcut bir sınıfın özelliklerinin ve davranışlarının başka bir sınıf tarafından devralınması prensibine dayanır. Bu yapı sayesinde sınıflar arasında hiyerarşik bir ilişki kurulur ve ortak özelliklerin tekrar tekrar yazılması önlenir. Üst sınıf genel özellikleri içerirken alt sınıflar daha özel davranışlar ekleyebilir veya var olan davranışları değiştirebilir. Kalıtım mekanizması kod tekrarını azaltması açısından önemli bir avantaj sağlar ve yazılım sistemlerinin daha düzenli bir mimari ile geliştirilmesine yardımcı olur. Gerçek dünya modellemesinde de kalıtım oldukça doğal bir yaklaşımdır; örneğin hayvan kavramı genel bir üst sınıf olarak düşünülebilirken kuş veya memeli gibi daha özel sınıflar bu yapıyı genişletebilir. Ancak kalıtımın bilinçli kullanılması gerekir. Gereğinden fazla derin kalıtım yapıları kodun karmaşıklığını artırabilir.



Nesneye yönelik programlamanın merkezinde sınıf ve nesne kavramları bulunmaktadır. Sınıf, bir nesnenin sahip olacağı özellikleri ve davranışları tanımlayan mantıksal bir şablondur. Tek başına çalıştırılabilir bir varlık değildir ve bellekte yer kaplamaz. Nesne ise sınıfın çalışma zamanında oluşturulmuş somut bir örneğidir ve bellekte gerçek bir yer kaplar. Aynı sınıftan birçok nesne üretilebilir. Bu nedenle sınıf bir plan veya tasarım olarak düşünülebilirken nesne bu planın gerçek dünyadaki uygulamasıdır. Örneğin “Araba” bir sınıf olabilirken farklı renk ve özelliklerdeki her araç ayrı bir nesne olarak düşünülebilir. Bu ayrımın anlaşılması nesneye yönelik düşünme biçiminin temelini oluşturur. Yazılım geliştirme sürecinde sınıflar sistemin genel yapısını ifade ederken nesneler programın çalışma anındaki dinamik davranışını temsil eder.