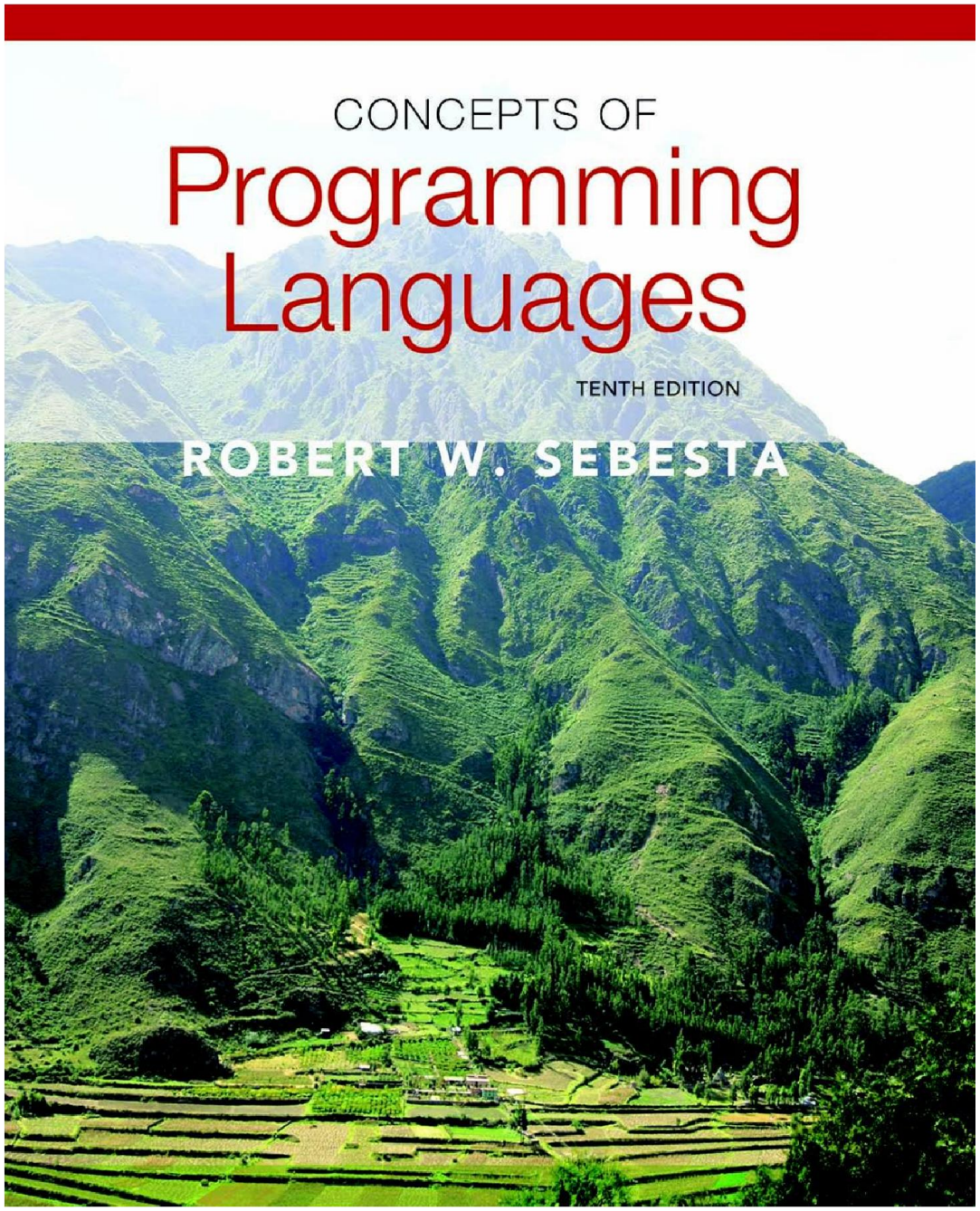
Bölüm 12

Nesne yönelimli programlama desteği



12. Bölüm konuları

- Giriş
- Nesne yönelimli programlama
- Nesne yönelimli diller için tasarım konuları
- C++ dilinde nesne yönelimli programlama desteği
- Nesne yönelimli programlama yapılarının gerçeklenmesi

Giriş

- Çok sayıda nesne yönelimli programlama (NYP) dili var:
  - Bazıları prosedürel programlamayı destekler (ör: Ada 95+ ve C++)
  - Bazları fonksiyonel programlamayı destekler (ör: CLOS – Common Lisp Object System)
  - Yeni diller (ör: Java ve C#) başka paradigmaları desteklemez ama onların komutsal yapılarını kullanır
  - Bazıları saf NYP dilidir (ör: Smalltalk, Ruby)

Nesne yönelimli programlama

- Üç ana özellik gerektirir:
  - Soyut veri tipleri
  - Kalıtım
    - NYP'nin ana teması
  - Polimorfizm

Kalıtım

- Tekrar kullanımdan kaynaklanan verimlilik artışı sağlar
- SVT'lerin özelleştirlerek tekrar kullanımı kod değişikliği gerektirir ve zahmetlidir
- Kalıtım, yeni sınıfları varolan sınıflara dayalı olarak tanımlar
- Varolan sınıfların hem yapısı, hem de işlevselliği kalıtım yolu ile alınabilir, ve yeni sınıf içinde değiştirilebilir
- Tekli – çoklu kalıtım

Nesne yönelimli kavramlar

- SVT'ler *sınıflar (classes)* aracılığı ile gerçeklenir
- Sınıf örneklerine *nesne (object)* denir
- Kalıtım yoluyla tanımlanan bir sınıfa *türemiş sınıf (derived class)* veya *altsınıf (subclass)* denir
- Kendisinden türeyen sınıf olan sınıfa *baba (parent)* sınıf veya *üstsınıf (superclass)* denir
- Nesneler üzerinde işlemler tanımlayan altprogramlara *metod* (*methods)* denir

Nesne yönelimli kavramlar...

- Metod çağırmaya *mesaj göderme* denir
- Bir nesnenin tüm metodlarının tümüne birden o nesnenin *mesaj protokolü (message protocol)* veya *mesaj arayüzü (message interface)* denir.
- Mesajların iki kısmı olur: metod ismi ve mesajin gideceği hedef nesne
- En basit durumda, bir sınıf babasının tüm özelliklerini kalıtım yolu ile alır

Nesne yönelimli kavramlar...

- Kalıtım, erişim kontrolleri nedeniyle karmaşık hale gelebilir
  - Sınıf, içindeki varlıkları altsınıflardan saklayabilir
  - Sınıf, içindeki varlıkları kullanıcılardan (clients) saklayabilir
  - Sınıf, içindeki varlıkları altsınıflara gösterebilir ama kullanıcılardan saklayabilir
- Sınıflar, kalıtımla aldıkları metodları değiştirebilirler
  - Yeni metod eskisini *hükümsüz kılar (overrides*)

Nesne yönelimli kavramlar...

- Bir altsınıf üç şekilde babasından farklılık gösterebilir:
  1. Üstsınıftaki *özel (private)* değişken veya metodları göremez
  2. Altsınıf, üstsınıfta olmayan değişken ve/veya metodlar ekleyebilir
  3. Altsınıf, kalıtımla aldığı metodların davranışını değiştirebilir.

Nesne yönelimli kavramlar...

- Sınıf içinde iki tür değişken var:
  - *Sınıf değişkenleri (class variables)* – bu tür değişkenlere bir kez yer ayrılır
  - *Örnek değişkenleri (instance variables)* – her nesne içinde yer ayrılır
- Sınıf içinde iki tür metod var:
  - *Sınıf metodu (class method)* – sınıfa gelen mesajları kabul eder
  - *Örnek metodu (instance method)* – nesneye gelen mesajları kabul eder

Sınıf metodları

- Sınıfa gelen mesajları kabul eder
- Normal fonksiyonlar gibidirler, ancak sınıfa aittirler.
- Sınıf dışından *SınıfAdı.metodAdı(...)* olarak çağrılırlar
- Sınıf içinden sadece *metodAdı(...)* olarak çağrılırlar

- (Sentaks bir dilden diğerine değişebilir)

Örnek metodları

- Nesneye gelen mesajları kabul eder
- Derleyici tarafından, fazladan bir parametre alan fonksiyona dönüştürülür. Bu fazladan parametre, mesajı alan nesneyi gösteren bir işaretçidir.
  - Örneğin *m,* C sınıfı içinde *void m(int a){…}* olarak tanımlı bir örnek metodu olsun.
  - *obj* bir C sınıfı örneği olsun
  - Derlerici bu tanımı *void m(C * this, int a){….}* şeklinde değiştirir
  - *obj.m(5)* ise *m(&obj, 5)* şekline dönüştütülür.

Dinamik bağlama

- *Polimorfik değişken* (*polymorphic variable)* bir sınıfın örneklerine ve o sınıfın atası olduğu (o sınıftan inen) tüm sınıflarının örneklerine işaret edebilir
- Polimorfik değişken aracılığı ile erişilen nesnelerde metod bağlaması dinamik olur (nesnenin sınıfına ait olan metod kullanılır; eğer yoksa, “en yakın” kalıtımla elde edilen metod çağrılır)

Soyut sınıflar ve metodlar

- *Soyut metod (abstract method)* tanımı olmayan metoddur (sadece protokolü tanımlar)
- *Soyut sınıf (abstract class)* en az bir tane soyut metodu olan sınıftır
- Soyut sınıfın örneği yaratılamaz

NYP dilleri tasarım problemleri

- Sadece nesneler mi olmalı?
- Altsınıflar alttipler mi?
- Tekli-çoklü kalıtım
- Nesne için yer alınması ve geri verilmesi
- Dinamik/Statik bağlama
- İçiçe sınıflar
- Nesnelerin ilklenmesi

Sadece nesnelerin olması

- Herşey nesne (ör. integer bile)
  - Avantajı – zarafet ve saflık
  - Dezavantajı– basit nesneler üzerindeki işlemeler yavaş
  - Örnek dil: Smalltalk
- Varolan tip sistemine nesneler eklenmesi
  - Avantajı – basit nesneler üzerindeki işlemeler hızlı
  - Dezavantajı – kafa karıştıran tip sistemi (iki tür varlık: nesneler ve kayıtlar (recordlar))
  - Örnek dil: C++

Sadece nesnelerin olması…

- Basit tipler için komutlu diller tarzında tip sistemi kullan, onların dışındaki herşey nesne olsun
  - Avantajları –
    - basit nesneler üzerindeki işlemeler hızlı
    - Nispeten küçük bir tip sistemi
  - Dezavantajı – yine kafa karıştıran tip sistemi
  - Örnek dil: JAVA

Altsınıflar altipler midir?

- Alt sınıfa ait bir nesneyi bir üst sınıfa ait diye düşünebilir miyiz?
- Öyle ise, altsınıf sadece yeni değişkenler ve metodlar ekleyebilmeli, kalıtım yolu ile aldığı metodları da ancak uyumlu bir şekilde değiştirmeli

Tekli ve çoklu kalıtım

- Çoklu kalıtım: birden çok sınıftan kalıtım alma (inherit)
- Çoklu kalıtımın dezavantajları
  - Dil ve implementasyon karmaşıklığı (kısmen isim çakışmalarından dolayı)
  - Olası verimsizlik – dinamik bağlama maliyeti daha fazla
- Avantajı:
  - Bazen işleri kolaylaştırıyor

Dinamik ve statik bağlama

- Mesajların hepsinin metodlara bağlanması dinamik olmalı mı?
  - Dinamik bağlama hiç yoksa, dinamik bağlamanın avantajları yok olur
  - Hepsi dinamikse, verimlilikten kaybeder (ör: Smalltalk, JAVA)
- Kullanıcıya kalmış (C++)

Smalltalk dilinde NYP desteği

- Smalltalk saf bir NYP dilidir
  - Herşey nesne
  - Her nesnenin yerel hafızası var
  - Tüm hesaplama nesnelerin birbirine mesaj göndermesiyle olur
  - Komutlu dillere hiç benzemez
  - Tüm nesneler yığın üzerinde yaratılırlar
  - Alınan yerin geri verilmesi (deallocation) otomatik olarak gerçekleşir

Smalltalk dilinde NYP desteği...

- Kalıtım
  - Altsınıf, üstsınıfın tüm örnek değişkenlerini, örnek metodlarını ve sınıf metodarını kalıtımla alır
  - Tüm altsınıflar alttiplerdir (hiçbirşey altsınıftan saklanamaz)
  - Sadece tekli kalıtım

Smalltalk dilinde NYP desteği...

- Dinamik bağlama
  - Mesajların metodlara bağlanması tümüyle dinamik
    - Mesajı alan nesnenin sınıfından başlayarak yukarıya doğru metodu ara
  - Dinamik tip kontrolü var
  - Tek “tip hatası”: mesaj anlaşılmadı

Smalltalk dilinde NYP desteği...

- Smalltalk değerlendirmesi
  - Dilin sentaksı basit, yalın ve düzgündür
  - Küçük bir dilin ne kadar güçlü olabileceğine örnek
  - Derlenmiş dillere göre daha yavaş
  - Dinamik bağlamadan dolayı tip hatalarının keşfedilmesi çalışma zamanına kalır
  - Grafik arayüz kullanımında öncü
  - En büyük etkisi: NYP'nin gelişmesi ve ilerlemesi

C++ dilinde NYP desteği

- Genel özellikleri:
  - C ve SIMULA 67 dillerinden evrimleşti
  - En çok kullanılan NYP dilleri arasında
  - Karışık (mixed) tip sistemi
  - Yapıcılar (constructors) ve yıkıcılar (destructors) var
  - Sınıf varlıklarına detaylı erişim kontolleri

C++ dilinde NYP desteği...

- Kalıtım
  - Bir sınıf herhangi başka bir sınıfın alt sınıfı olmak zorunda değil
  - Erişim kontrol komutları
    - Private (sadece sınıf içinde ve dostlar tarafından görünebilir) (altsınıfların alttip olmasını engeller)
    - Public (altsınıflarda ve kullanıcılar (clients) tarafından görülebilir)
    - Protected (sınıf içinde ve altsınıflarda görülebilir, ama kullanıcılar (clients) tarafından görülemez)

C++ dilinde NYP desteği...

- Çoklu kalıtım desteklenir
  - Ayni isimde iki tane kalıtımla gelmiş üye varsa, istenilene `::` operatörü ile erişilebilir

```
class Thread { ... }
class Drawing { ... }
class DrawThread : public Thread, public Drawing
  { … }
```

C++ dilinde NYP desteği...

- Dinamik bağlama
  - `virtual` olarak tanımlanan metodlar polimorfik değişkenler aracılığı ile çağrılabilirler ve metodlara dinamik olarak bağlanırlar
  - "pure virtual" olarak tanımlanmış bir metodun gövdesi yoktur (yani soyuttur)
  - İçinde en a bir tane "pure virtual" metod olan sınıf, *sanal* bir sınıftır *(abstract class)*

C++ dilinde NYP desteği...

- Değerlendirme
  - C++ dilinde bol miktarda erişim kontrolü vardır
  - C++ dilinde çoklu kalıtım vardır
  - C++ dilinde programcı tasarım zamanında hangi metodun dinamik, hangisinin statik olarak bağlanacağını belirtmelidir
    - Statik bağlama daha hızlı!
  - Tercüme (interpretation) ve dinamik bağlamadan dolayı Smalltalk C++ya göre 10 misli kadar daha yavaş

Nesne yönelimli yapıların gerçeklenmesi

- İki önemli konu
  - Örnek değişkenlerinin (instance variables) hafızadaki yapıları
  - Mesajların metodlara dinamik bağlanması

Örneklerin veri saklaması

- Sınıf Örnek Kayıtları (SÖK) (Class Instance Records) nesnenin durumunu saklar
  - Statikdir (derleme zamanında belirlenir)
- Bir sınıfın babası varsa, çocuğun örnek değişkenleri babanın sınıf örnek kaydına eklenir
- Mesajların metodlara dinamik bağlanması destekleniyorsa, SÖK içinde sınıfın Sanal Metod Tablosu'na (SMT) (Virtual Method Table (VMT)) bir işaretçi olması gerekir

Metod çağırmalarının dinamik bağlanması

- Sanal Metod Tablosu (SMT) (Virtual Method Table): Her sınıf için bir tane olmak üzere, sınıfın örnekleri aracılığı ile çağrılabilen metodların kodlarına işaretçileri barındıran tablo
- Sadece dinamik olarak bağlanan metodların SMT'de girdisi olması gereklidir (statik olarak bağlanan metodların SMT'de olmaları gerekmez)
- Dinamik olarak bağlanan metodlara çağrılar, metodun koduna SÖK içindeki SMT işaretçisi aracılığı ile olur

Metod çağırmalarının dinamik bağlanması...

- Metod çağrıları VMT başlangıcından itibaren göreceli adres olarak temsil edilebilir
- Baba-çocuk ilişkisi içinde olan iki sınıfa ait olan bir metodun her iki sınıfın SMT'si içindeki göreceli adresi ayni olur
- Çocuk sınıfta yeni olarak tanımlanan metodlar, babanın SMT'sinin sonuna eklenir
- e.f() çağrıldığı zaman olanlar (e'nin sınıfı E ise) : e'nin SÖK'ü içinde E sınıfının SMT'sinin adresi bulunur. f() metodunun göreceli adresi 10 ise, f()'in kodunun adresi, SMTnin adresi + 10 dur. f()'in koduna gidilir (fonksiyon çağırması olarak)

Özet

- NYP'de üç önemli kavram var: Soyut Veri Tipleri, kalıtım, dinamik bağlama
- Ana tasarım problemleri: sadece nesneler mi var? Altsınıflar alttip midirler? Polymorfizm varmıdır? Tekli-Çoklu kalıtım? Dinamik bağlama?
- Smalltalk saf bir nesne yönelimli dildir
- C++ dilinde iki farklı tip sistemi mevcuttur (hibrid)
- NYP gerçeklenmesi için yeni veri yapılarına ihtiyaç vardır (SMT vs.)