



BMB214 Programlama Dilleri Prensipleri

Ders 11. Soyut Veri Türleri (Abstract Data Types) ve Kapsülleme (Encapsulation) Kavramları

Konular

- ◎ Soyutlama (Abstraction) Kavramı
- ◎ Veri Soyutlamasına (Data Abstraction) Giriş
- ◎ Soyut Veri Türleri (Abstract Data Types) için Tasarım Sorunları
- ◎ Dil Örnekleri
- ◎ Parametrelili (Parameterized) Soyut Veri Türleri
- ◎ Kapsülleme (Encapsulation) Yapıları
- ◎ Kapsüllemeleri Adlandırma

Soyutlama (Abstraction) Kavramı

- ◎ Bir soyutlama, yalnızca en önemli nitelikleri içeren bir varlığın görünümü veya temsilidir
- ◎ Soyutlama programlamada (ve bilgisayar biliminde) temel bir kavramdır
- ◎ Neredeyse tüm programlama dilleri, alt programlarla (subprograms) işlem soyutlamayı (process abstraction) destekler
- ◎ 1980'den beri tasarlanan neredeyse tüm programlama dilleri (data abstraction) veri soyutlamayı destekler

Veri Soyutlamasına (Data Abstraction) Giriş

- ◎ Soyut bir veri türü (abstract data type), aşağıdaki iki koşulu karşılayan kullanıcı tanımlı bir veri türüdür:
 - Türdeki nesnelerin temsili, bu nesneleri kullanan program birimlerinden gizlenmiştir, bu nedenle olası işlemler yalnızca türün tanımında sağlananlardır.
 - Türdeki nesneler üzerindeki işlemlerin türü ve protokollerinin bildirimleri tek bir sözdizimsel birimde bulunur. Diğer program birimlerinin tanımlanan tipte değişkenler oluşturmalarına izin verilir.

Veri Soyutlamanın Avantajları

⊙ İlk koşulun avantajları

- Güvenilirlik - veri temsillerini gizleyerek, kullanıcı kodu türdeki nesnelere doğrudan erişemez veya temsile bağlı olamaz, bu da temsilin kullanıcı kodunu etkilemeden değiştirilmesine izin verir.
- Programcının bilmesi gereken kod aralığını ve değişkenleri azaltır
- İsim uyuşmazlıkları daha az olasıdır

⊙ İkinci koşulun avantajları

- Bir program düzenleme yöntemi sağlar
- Değiştirilebilirliğe yardımcı olur (bir veri yapısıyla ilişkili her şey bir arada)
- Ayır derleme (compilation)

Soyut Veri Türleri için Dil Gereksimleri

- ◎ Tür tanımının kapsülleneceği sözdizimsel bir birim (syntactic unit)
- ◎ Gerçek tanımları gizlerken, tür adlarını ve alt program başlıklarını istemcilere görünür yapma yöntemi
- ◎ Dil işlemciye (language processor) bazı ilkel işlemler yerleştirilmelidir

Tasarım Zorlukları

- ◎ Soyut türler parametreleştirilebilir mi?
- ◎ Hangi erişim kontrolleri (access controls) sağlanmıştır?
- ◎ Türün özellikleri, uygulamasından fiziksel olarak ayrı mı?

Dil Örnekleri

C ++

- ◎ C **struct** türüne ve Simula 67 sınıfların (classes)
- ◎ Sınıf (class), kapsülleme (encapsulation) için kullanılır
- ◎ Bir sınıf bir türdür
- ◎ Bir sınıfın tüm sınıf örnekleri (class instances), üye fonksiyonların tek bir kopyasını paylaşır
- ◎ Bir sınıfın her örneği, sınıf veri üyelerinin kendi kopyasına sahiptir
- ◎ Örnekler statik, stack dynamic veya heap dynamic olabilir

Dil Örnekleri

C ++

- ◎ Bilgi gizleme (Information Hiding)
 - Gizli varlıklar için private clause
 - Arayüz varlıkları için public clause
 - Kalıtım (Inheritance) için protected clause

Dil Örnekleri

C ++

◎ Yapıcılar (Constructor):

- Örneklerin (instances) veri üyelerini başlatmak için fonksiyon/lar (nesneleri oluşturmazlar)
- Nesnenin bir kısmı heap-dynamic ise depolama da tahsis edebilir
- Nesnelerin parametrelendirilmesini sağlamak için parametreler içerebilir
- Bir örnek oluşturulduğunda örtülü (implicitly) olarak çağrılır
- Açıkça (explicitly) çağrılabilir
- Tanımlaması, sınıf adıyla aynıdır

Dil Örnekleri

C ++

◎ Yıkıcılar (Destructors)

- Bir örnek yok edildikten sonra çalışacak son fonksiyon; genellikle sadece heap depolamayı geri kazanmak için
- Nesnenin ömrü sona erdiğinde örtülü (implicitly) olarak çağrılır
- Açıkça (explicitly) çağrılabilir
- Tanımlamasında, önünde tilde (~) bulunan sınıf adıdır

Dil Örnekleri

C++

```
class Stack {  
    private:  
        int *stackPtr, maxLen, topPtr;  
    public:  
        Stack() { //constructor  
            stackPtr = new int [100];  
            maxLen = 99;  
            topPtr = -1;  
        };  
        ~Stack () {delete [] stackPtr;}; //destructor  
        void push (int number) {  
            if (topSub == maxLen)  
                cerr << "push Hatası - stack dolu\n";  
            else stackPtr[++topSub] = number;  
        };  
        void pop () {...};  
        int top () {...};  
        int empty () {...};  
}
```

Dil Örnekleri

C ++ Header Dosyası

```
// Stack.h - Stack sınıfı için header dosyası
#include <iostream.h>
class Stack {
private: /** Bu üyeler sadece sınıf içinden görülür
/** üyeler
    int *stackPtr;
    int maxLen;
    int topPtr;
public: /** Bu üyeler istemciden (object, instance) görülebilir
    Stack(); /** A constructor
    ~Stack(); /** A destructor
    void push(int);
    void pop();
    int top();
    int empty();
}
```

Dil Örnekleri

C++ Stack Sınıfı

```
// Stack.cpp - Stack sınıfı için uygulama
#include <iostream.h>
#include "Stack.h"
using std::cout;
Stack::Stack() { //constructor
    stackPtr = new int [100];
    maxLen = 99;
    topPtr = -1;
}
Stack::~Stack() {delete [] stackPtr;}; //destructor
void Stack::push(int number) {
    if (topPtr == maxLen)
        cerr << "push Hatası - stack dolu\n";
    else stackPtr[++topPtr] = number;
}
...
```

Dil Örnekleri

C ++

- ◎ Arkadaş fonksiyonları veya sınıfları (Friend functions or classes) - bazı ilgisiz birimlere veya fonksiyonlara özel üyelere erişim sağlamak için kullanılır
 - C++'da gerekli

Dil Örnekleri

Java

- ◎ Aşağıdakiler dışında C++'ya benzer:
 - Tüm kullanıcı tanımlı türler (user-defined types) sınıflardır
 - Tüm nesneler (objects) heap'ten tahsis edilir ve referans değişkenleri aracılığıyla erişilir
 - Sınıflardaki bireysel varlıklar, clause yerine erişim denetimi değiştiricilere (public veya private) sahiptir
 - Tüm nesnelerin örtülü çöp toplama (implicit garbage collection - GC)
 - Java, arkadaşların yerine kullanılabilecek ikinci bir kapsam mekanizmasına, paket kapsamına sahiptir.
 - Bir paketteki, erişim belirteçlerine(control modifiers) sahip olmayan tüm sınıflardaki varlıklar, paket boyunca görülebilir

Dil Örnekleri

Java

```
class StackClass {  
    private:  
        private int [] *stackRef;  
        private int [] maxLen, topIndex;  
        public StackClass() { // a constructor  
            stackRef = new int [100];  
            maxLen = 99;  
            topPtr = -1;  
        };  
        public void push (int num) {...};  
        public void pop () {...};  
        public int top () {...};  
        public boolean empty () {...};  
}
```

Dil Örnekleri

C#

- ◎ C++ ve Java'yı temel alır
- ◎ internal and protected internal olmak üzere iki erişim belirteci (access modifier) ekler
- ◎ Tüm sınıf örnekleri (instances) heap dynamic'tir.
- ◎ Varsayılan yapıcılar (Default constructors) tüm sınıflar için mevcuttur
- ◎ Çöp toplama (Garbage collection – GC) çoğu heap nesnesi için kullanılır, bu nedenle yıkıcılar (destructors) nadiren kullanılır
- ◎ struct'lar, kalıtımı desteklemeyen hafif (lightweight) sınıflardır

Dil Örnekleri

C#

- ⦿ Veri üyelerine (data members) erişim ihtiyacı için ortak çözüm: erişimci yöntemleri (accessor methods) getter ve setter'dır
- ⦿ C #, açık (explicit) metot çağrıları gerektirmeden getters and setters'lara uygulamanın bir yolu olarak property sağlar

Dil Örnekler

C# Property Örneği

```
public class Weather {  
    public int DegreeDays { /** DegreeDays bir property'dir  
        get {return degreeDays;}  
        set {  
            if (value < 0 || value > 30)  
                Console.WriteLine(«Değer aralık dışında: {0}», value);  
            else degreeDays = value;}  
        }  
    private int degreeDays;  
    ...  
}  
...  
Weather w = new Weather();  
int degreeDaysToday, oldDegreeDays;  
...  
w.DegreeDays = degreeDaysToday;  
...  
oldDegreeDays = w.DegreeDays;
```

Soyut Veri Türleri (Abstract Data Types)

Ruby

- ⦿ Kapsülleme yapıcı (Encapsulation construct) sınıfıdır
- ⦿ Yerel değişkenlerin "normal" adları vardır
- ⦿ Örnek değişken adları "@" işaretleriyle başlar
- ⦿ Sınıf değişkeni adları "@@" işaretiyle başlar
- ⦿ Örnek metotları (Instance methods) Ruby fonksiyonlarının sözdizimine sahiptir (def... end)
- ⦿ Constructor'lar adı initialize (sınıf başına yalnızca bir tane) —new çağrıldığında açıkça (implicitly) çağrılır
 - Daha fazla constructors ihtiyaç duyulursa, farklı adlara sahip olmalı ve açıkça new ile çağırılmalıdır.
- ⦿ Sınıf üyeleri (Class members) private veya public olarak işaretlenebilir ve varsayılan olarak genel olarak işaretlenebilir
- ⦿ Sınıflar dinamiktir

Örnek Kod

Ruby

```
class StackClass {  
  def initialize  
    @stackRef = Array.new  
    @maxLen = 100  
    @topIndex = -1  
  end  
  
  def push(number)  
    if @topIndex == @maxLen  
      puts "Error in push - stack is full"  
    else  
      @topIndex = @topIndex + 1  
      @stackRef[@topIndex] = number  
    end  
  end  
  
  def pop ... end  
  def top ... end  
  def empty ... end  
end
```

Parametrelili Soyut Veri Türleri (Parameterized Abstract Data Types)

- ◎ Parametrelili soyut veri türleri, herhangi bir tür ögesini depolayabilen bir soyut veri türünün tasarlanmasına izin verir
 - Yalnızca statik yazılan diller için bir sorun vardır
- ◎ Generic sınıflar (generic classes) olarak da bilinir
- ◎ C ++, Java 5.0 ve C # 2005, parametrelili soyut veri türleri için destek sağlar

Parametrelili Soyut Veri Türleri C++

- ◎ Sınıflar, parametreleştirilmiş yapıcı fonksiyon yazarak generic olabilir

```
Stack (int size) {  
    stk_ptr = new int [size];  
    max_len = size - 1;  
    top = -1;  
};
```

- ◎ Stack nesnesinin bildirimi

```
Stack stk(150);
```


Parametrelili Soyut Veri Türleri

C++

◎ Stack öge türü, template bir sınıf yaparak parametrelendirilebilir

```
template <class Type>
class Stack {
private:
    Type *stackPtr;
    const int maxLen;
    int topPtr;
public:
    Stack() { // Constructor for 100 elements
        stackPtr = new Type[100];
        maxLen = 99;
        topPtr = -1;
    }

    Stack(int size) { // Constructor for a given number
        stackPtr = new Type[size];
        maxLen = size - 1;
        topSub = -1;
    }
    ...
}
```

◎ Tanımlama

```
Stack<int> myIntStack;
```

Parametrelili Sınıflar

Java 5.0

- ⦿ Generic parametreler sınıflar olmalıdır
- ⦿ En yaygın generic türler, LinkedList ve ArrayList gibi koleksiyon türleridir.
- ⦿ Kullanıcılar generic sınıfları tanımlayabilir
- ⦿ Generic koleksiyon sınıfları (Generic collection classes) ilkeleri depolayamaz
- ⦿ İndeksleme desteklenmiyor
- ⦿ Önceden tanımlanmış bir generic sınıfın kullanımına örnek:

```
ArrayList <Integer> myArray = new ArrayList <Integer> ();  
myArray.add(0, 47); // 1 eleman ekle...
```

Parametrelili Sınıflar

Java 5.0

```
import java.util.*;

public class Stack2<T> {
    private ArrayList<T> stackRef;
    private int maxLen;
    public Stack2() {
        stackRef = new ArrayList<T> ();
        maxLen = 99;
    }
    public void push(T newValue) {
        if (stackRef.size() == maxLen)
            System.out.println("push hatası - stack dolu");
        else
            stackRef.add(newValue);
        ...
    }
}

Tanımlama: Stack2<string> myStack = new Stack2<string> ();
```

Parametrelı Sınıflar

C#

- ⦿ Joker karakter sınıflarının (wildcard classes) olmaması dışında Java 5.0'dakilere benzer
- ⦿ Array, List, Stack, Queue ve Dictionary için önceden tanımlanmıştır
- ⦿ Parametrelı yapıların elemanlarına indeksleme yoluyla erişilebilir

Encapsulation Yapıları

- ◎ Büyük programların iki özel ihtiyacı vardır:
 - Basitçe alt programlara bölmek dışında bazı organizasyon araçları
 - Bazı kısmi derleme yöntemleri (tüm programdan daha küçük olan derleme birimleri)
- ◎ Çözüm: Ayrı ayrı derlenebilen bir birimle mantıksal olarak ilişkili olan bir alt program grubu (derleme birimleri - compilation units)
- ◎ Bu tür koleksiyonlara kapsülleme denir

İç içe geçmiş alt programlar (Nested Subprograms)

- ⦿ Alt program tanımlarını, onları kullanan mantıksal olarak daha büyük alt programların içine yerleştirerek programları düzenleme
- ⦿ İç içe geçmiş alt programlar Python, JavaScript ve Ruby'de desteklenir

Kapsülleme C

- ◎ Bir veya daha fazla alt program içeren dosyalar bağımsız olarak derlenebilir
- ◎ Arayüz bir header dosyasına yerleştirilir
 - Sorun 1: Bağlayıcı (linker), bir header ve ilişkili uygulama arasındaki türleri kontrol etmiyor
 - Sorun 2: İşaretçilerle ilgili sorunlar
- ◎ #include preprocessor specification - uygulamalara header dosyalarını dahil etmek için kullanılır

Kapsülleme

C++

- ◎ C'ye benzer header ve kod dosyalarını tanımlayabilir
- ◎ Veya sınıflar kapsülleme için kullanılabilir
 - Sınıf, arayüz (interface) olarak kullanılır (prototypes)
 - Üye tanımları ayrı bir dosyada tanımlanır
- ◎ Friends, bir sınıfın private üyelerine erişim sağlamanın bir yolunu sağlar

C# Assemblies

- ◎ Uygulama programlarına tek bir dinamik bağlantı kitaplığı (dynamic link library - DLL) veya yürütülebilir olarak görünen bir dosya koleksiyonu
- ◎ Her dosya, ayrı ayrı derlenebilen bir modül içerir.
- ◎ Bir DLL, çalışan bir programa ayrı ayrı bağlanan sınıflar ve metotlar koleksiyonudur.
- ◎ C#, internal adında bir erişim belirteci (access modifier) sahiptir; bir sınıfın internal bir üyesi, görüldüğü derlemedeki tüm sınıflar tarafından görülebilir

Kapsüllemeleri Adlandırma (Naming)

- ◎ Büyük programlar birçok global adı tanımlar; mantıksal gruplara bölmek için bir yola ihtiyaç var
- ◎ Adlar için yeni bir kapsam oluşturmak için bir kapsülleme adlandırması kullanılır
- ◎ C++ Namespaces
 - Her kitaplığı kendi namespace yerleştirebilir ve namespace dışında kullanılan adları niteleyebilir
 - C# ayrıca namespace de içerir

Kapsüllemeleri Adlandırma...

◎ Java Paketleri (Packages)

- Paketler birden fazla sınıf tanımlı içerebilir; bir paketteki sınıflar kısmi arkadaşlardır (friends)
- Bir paketin istemcileri tam nitelikli ad kullanabilir veya import anahtar kelimesi kullanabilir

Kapsüllemeleri Adlandırma...

◎ Ruby Modülleri (Modules):

- Ruby sınıfları isim kapsüllemesidir, ancak Ruby'de modüller de vardır
- Tipik olarak sabitlerin ve yöntemlerin koleksiyonlarını kapsüllenir
- Modüller somutlaştırılmaz veya alt sınıflara alınamaz ve değişkenleri tanımlayamazlar
- Bir modülde tanımlanan yöntemler, modülün adını içermelidir
- require metodu ile bir modülün içeriğine erişim talep edilir