Programlama Dillerinin Prensipleri

Lab Notları – 6

Modüler Programlama ve Fonksiyonlar

Modül (Alt program- subprogram): Oluşturulmuş olan bir kod bloğunu birçok yerde kullanmak. Böylelikle kullanılan her yerde tekrar tekrar yazmaktan kurtulmuş oluruz. İşte bu kod bloğu alt program yani modüldür. Bellekten ve fazla kod yazımından kazanılmış olur. Oluşturulan bir kod bloğunun birçok yerde kullanımından kasıt, o kod bloğunu kullanılan her yerde çağırmak. Örnek olarak bir metot veya fonksiyon çağrımı düşünülebilir.

Her alt programın bir giriş noktası vardır. Fakat birçok farklı yönden bir alt program dönebilir. Bir fonksiyon sadece ona verilen görevi yapmalıdır.

Bir fonksiyon tanımlaması C ve java için aşağıdaki gibidir.

```
dönüş_değeri    fonksiyon_adi(1. parametre, 2. Parametre,…)
{
        // Fonksiyonun gövdesi
}
```

Java'da metot ve fonksiyonlar mutlaka sınıf içerisinde tanımlanmalıdırlar. Sınıf içinde yazılan fonksiyonlara üye fonksiyonlar denir. Aşağıda Java'da basit bir Sayı sınıfı ve bu sınıfa ait metot tanımlanmıştır.

```
public class Sayi {
   private int deger;
   private int uzunluk;
   public Sayi(int dgr){
      deger=dgr;
      uzunluk = String.valueOf(deger).length();
   }
   public short[] Rakamlar(){
      int tmp = deger;
      short []rakamlar = new short[uzunluk];
      int indeks=uzunluk-1;
      while(tmp > 0) {
         rakamlar[indeks--] = (short)(tmp % 10);
         tmp /= 10;
      }
      return rakamlar;
   }
   public int Uzunluk(){
      return uzunluk;
   }
}
```

```
public static void main(String[] args) {
      // TODO code application logic here
      Scanner ekran = new Scanner(System.in);
      System.out.print("Tam Sayı:");
```

```
        int sayi = ekran.nextInt();
        Sayi s = new Sayi(sayi);
        short []rakamlar = s.Rakamlar();
        for(int indeks=0;indeks<s.Uzunluk();indeks++){
           System.out.print(rakamlar[indeks]+" ");
        }
    }
```

Prototip Tanımlama: C ve C++ dilinde metot ve fonksiyonlar değişkenler gibidir. Çağrıldıkları yerden daha yukarıda tanımlanmış olmaları gerekir. Bu tanımlama metodun tamamı olabileceği gibi sadece değişken tanımı gibi tanımlama yapılabilir. Bu tanımlama işlemine **prototip** tanımlama denir. Prototip tanımlamada metodun içeriği yazılmaz sadece parametre türleri ve dönüş türü yazılır. Örneğin aşağıdaki kod derlenmeye çalışıldığında hata verecektir. Çünkü Topla fonksiyonu çağrıldığı yerden daha aşağıda tanımlanmıştır. Bunu çözmek için ya Topla fonksiyonu çağrıldığı yerin yukarısına alınmalı ya da Topla fonksiyonunun prototipi yukarıda tanımlanmalıdır.

```
// Hatalı Program
#include "stdio.h"

int main(){
        printf("%lf\n",Topla(5.2,18.8));
        return 0;
}
double Topla(double a,double b){
        return a+b;
}
```

```
// Doğrusu
#include "stdio.h"
double Topla(double,double);

int main(){
        printf("%lf\n",Topla(5.2,18.8));
        return 0;
}
double Topla(double a,double b){
        return a+b;
}
```

Bunun dışında C dilinde, C++'tan farklı olarak dönüş türü int olan fonksiyonların prototipi tanımlanması zorunlu değildir. Derleyici bir fonksiyonu çağrıldığı yerden daha önce bulamaz ise onun dönüş türünün int olduğunu varsayacak ve dosyada bu fonksiyonu arayacaktır. Aşağıdaki kodta C derleyicisi hata vermeyecektir.

```
// Hata Vermeyen Program
#include "stdio.h"
int main(){
        printf("%d\n",Topla(5,18));
        return 0;
}
int Topla(int a,int b){
        return a+b;
}
```

Parametreler:

C dili için alt programın (metot ya da fonksiyon) veriye erişmesinin iki yolu vardır. Bunlardan biri lokal olmayan değişken tanımlayıp erişmek diğeri ise parametreler. Lokal olmayan değişken yani global değişken tanımı önerilmeyen bir yöntemdir. Dolayısıyla en iyi yol parametre ile alt yordamların iletişime geçmesidir. Parametreler lokal değişkenlerdir ve çalışma anı yığınında tutulurlar. Fonksiyon çağrısı bittiğinde bellekten silinirler. Java'da ise sınıf dışında metot olmayacağı için metotlara

parametreler yardımıyla iletişime geçilmelidir. Aşağıda C dilinde kombinasyon hesabı yapan kod görülmektedir.

```
#include "stdio.h"
double Fak(int sayi){
        int i;
        double toplam=1;
        for(i=2;i<=sayi;i++) toplam*=i;
        return toplam;
}
double Komb(int x,int y){
        return Fak(x)) / (Fak(x-y)*Fak(y));
}
int main(){
        int x,y;
        printf("x:");
        scanf("%d",&x);
        printf("y:");
        scanf("%d",&y);
        printf("Komb(%d,%d)=%.2lf\n",x,y,Komb(x,y));
        return 0;
}
```

Parametre Geçirme Yöntemleri:

Pass-By-Value (Değer ile Çağırma): Çağıran, çağırılana değeri direk gönderir. Formal parametre, asıl parametre ile ilklenmiş olur. Örneğin aşağıda daha önce yazılmış olan Sayı sınıfına DegerAta adında bir metot tanımlanıyor.

```
Public class Sayi{
…
public void DegerAta(int yeniDeger){
      deger=yeniDeger;
   }
…
}
```

```
Sayi s = new Sayi(120);
int x =10;
s.DegerAta(x);
System.out.println(s.Deger());
```

Pass-By-Reference (Referans ile Çağırma): C dilinde desteklenmeyen bu çağırım şekli C++ dilinde desteklenmektedir. Java referans ile çağırma desteklenmez her türlü çağırım değer ile çağırmadır Aşağıdaki C++ kodu C'de yazılamaz,

```
// C dilinde Hata verir.
void degistir(int& x,int& y){
        int z=x;
        x=y;
        y=z;
}
```

Pass-By-Pointer (Gösterici ile Çağırma): Bu çağrım türünde parametre türü bir göstericidir ve çağırırken değer yerine adres gönderilir. Dolayısıyla adres ile çağırma olarak ta isimlendirilir. Aşağıdaki C kodunda değiştir fonksiyonu parametre olarak bir gösterici alır ve gösterdiği adresteki değeri 100 yapar. Fonksiyon çağrımı bittiğinde p'nin gösterdiği adreste dolayısıyla a değerinin içinde 100 yacaktır. Java dilinde de buna benzer bir çağırma yöntemi vardır. Aşağıdaki kod bloğunda Sayi sınıfının yapıcı metoduna gönderilen s nesneyi referans eden bir adrestir.

```c
#include "stdio.h"

void degistir(int *x){
        *x = 100;
}
int main(){
        int a=10;
        int *p = &a;
        printf("a:%d\n",*p);
        degistir(p);
        printf("a:%d\n",*p);
        return 0;
}
```

```java
//Java
class Sayi{
    ....
    public Sayi(Sayi s){
        this.deger = s.Deger();
        this.uzunluk = s.Uzunluk();
    }

}
```

Dizinin parametre ile gönderilmesi

```c
//Yanlış Kullanım
char* IsimAl(){
        char isim[50];
        printf("isim:");
        scanf("%s",isim);
        return isim;
}
int main(){
        char *str = IsimAl();
        printf("%s\n",str);
        return 0;
}
```

```c
// Doğru kullanım
#include "stdio.h"
#include "stdlib.h"

char* IsimAl(){
        char *isim = (char*)malloc(sizeof(char)*50);
        printf("isim:");
        scanf("%s",isim);
        return isim;
}
int main(){
        char *str = IsimAl();
        printf("%s\n",str);
        free(str);
        return 0;
}
```

C dilinde olmasa da C++ için referans ile çağırma tercih edilmelidir. Çünkü bir göstericinin NULL ifadeye atanma ihtimali vardır. Bir gösterici bellek adresi içerirken bir referans, temsil ettiği değer ile aynı bellek adresindedir. Gösterici ile çağırma kısmında p göstericisi NULL ifadesine atanmasına rağmen r göstericisini etkilememiştir.

```c
#include <iostream>
using namespace std;
void GostericiCagirma(int *p){
        p=NULL;
}
void ReferansCagirma(int &x){
```

```
        x=100;
}
int main(){
        int a=50;
        int *r=&a;
        cout<<"a:"<<*r;
        GostericiCagirma(r);
        cout<<endl<<"a:"<<*r;
        int b=50;
        cout<<endl<<"b:"<<b;
        ReferansCagirma(b);
        cout<<endl<<"b:"<<b;
        return 0;
}
```

C dilinde aşağıdaki örnekte hem gösterici ile hem de değer ile çağırma örneği bir arada kullanılmıştır. Parametre olarak verilen dizinin ilk adresi ve dizinin uzunluğu fonksiyon içerisinde kullanılıp dizi ters çevrilmiş ve main fonksiyonundaki parametre olarak gönderilen dizi değişkeni bundan etkilenmiş ve dizi ters çevrilmiştir.

```
#include "stdio.h"

void TersCevir(int *dizi, int uzunluk)
{
  int tmp; // değişme işlemi için
  int i;
  for (i = 0; i < uzunluk/2; i++)
  {
    tmp = dizi[uzunluk-i-1];
    dizi[uzunluk-i-1] = dizi[i];
    dizi[i] = tmp;
  }
}
int main(){
        int dizi[] = {15, 82, 16, 90, 2, 12, 100};
        TersCevir(dizi,7);
        int i;
        for(i=0;i<7;i++) printf("%d ",dizi[i]);
        return 0;
}
```

Pass-By-Result (Sonuç ile Çağırma): sonuç (out) parametresi alt programa (metot ya da fonksiyon) bir değer olarak gönderilmez bilakis fonksiyon içerisinde değer alıp gelir. Formal parametre lokal değişken gibi davranır ve çağırana değer olarak döner. C++'ta referans çağırma yöntemi kullanılarak yapılabilecek bu işlem C dilinde referans ile çağırma desteklenmediği için C dilinde sonuç ile çağırma yöntemi **uygulanamaz**

Özyineleme (Recursion)

Bir fonksiyon veya metodun kendi kendini çağırma işlemine özyineleme denir. Özyinelemenin 2 temel kuralı vardır.

- Sonlanacağı durum (Temel adım)
- Her çağrıda sonlanacağı duruma yaklaşması (Özyineleme)

Eğer bir sonlanma durumu olmaz ise sonsuz çağrım olacak ve program hata verecektir. Sonlanma durumu var ama her çağrıda o duruma yaklaşılmaz ise yine sonsuz çağrım olur. Özyineleme birçok problemde kod satırlarının, bir iki satıra inmesini sağlar. Fakat sürekli bellekte yeni çağrımlar için yer ayrılacak ve yavaşlamaya sebep olacaktır. Aşağıda daha önce Java'da yazılmış olan Sayı sınıfına Faktoriyel metodu ekleniyor bu metot Sayı nesnesinin içinde taşıdığı değerin faktöriyelini döndüren bir metot fakat bu metot Fakt metodunu içeriden çağırıyor ve Fakt metodu aslında özyineleme kullanarak sonucu bulan bir metot.

```
public class Sayi{
…
   private int Fakt(int sayi){
      if(sayi<=1) return 1;
      return sayi * Fakt(sayi-1);
   }
   public int Faktoriyel(){
      return Fakt(deger);
   }
…
```

```
public static void main(String[] args) {
      // TODO code application logic here
      Sayi s = new Sayi(5);
      System.out.println(s.Faktoriyel());
   }
```

Varsayılan Parametre: C ve Java dillerinde varsayılan parametre desteklenmez. C++'ta desteklenen bu özellik eğer parametreye bir değer girilmez ise varsayılan olarak verilmiş değeri alır.

Değişken Sayıda Parametre: Bir fonksiyonun değişken sayıda parametre alması, o fonksiyonun farklı sayıda parametre ile çağrılmasını sağlar. Bu işlem C ve Java'da desteklenmektedir.

```
public class Sayi{
…
  public boolean Asalmi(int... bolen){
      if(bolen.length == 0) return Asalmi(2);
      if(deger == bolen[0]) return true;
      if(deger%bolen[0] == 0) return false;
      return Asalmi(bolen[0]+1);
   }
…
}
```

```
public static void main(String[] args) {
      // TODO code application logic here
      Scanner ekran = new Scanner(System.in);
```

```
        System.out.print("Sayı:");
        int x = ekran.nextInt();
        Sayi s = new Sayi(x);
        if(s.Asalmi()) System.out.println("Girilen deger asal");
        else System.out.println("Girilen deger asal değil");
    }
```

C'deki kullanımı aşağıdaki gibidir. Fakat va_arg'ın kullanılabilmesi için koda "stdarg.h" kütüphanesinin eklenmesi gerekir.

```
#include "stdio.h"
#include "stdarg.h"
typedef enum BOOL{
        false, true
}bool;
bool Asal(int parametreAdedi,...)
{
  va_list valist; //parametre listesi tanımlanıyor.
  double sum = 0.0;
  // adet kadar parametre listeye atılıyor
  va_start(valist, parametreAdedi);
  int sayi = va_arg(valist, int);
  if(parametreAdedi == 1){
                va_end(valist);
                return Asal(2,sayi,2);
        }

  int i = va_arg(valist, int);
  va_end(valist);

  if(sayi == i) return true;
  if(sayi%i == 0) return false;
  return Asal(2,sayi,i+1);
}
int main(){
        int x;
        printf("x:");
        scanf("%d",&x);
        if(Asal(1,x)) printf("x sayisi asaldir.");
        else printf("x sayisi asal degildir.");
        return 0;
}
```

Sabit Fonksiyon Tanımlama

Java ve C için sabit tanımlamaları daha önce gösterilmişti. İlkel türler için C dili const ifadesi ile Java'da final ifadesi ile sabit tanımlanabiliyordu. C++ dilinde ise aynı durum fonksiyonlar içinde geçerlidir. Bir fonksiyon tanımının sonuna getirilen const ifadesi o fonksiyonu sabit yapar ve fonksiyon içinde değerin değiştirilmesine izin vermez. Fakat bunun Java ve C dilinde bir karşılığı yoktur.

Java'da Kendinize ait Kütüphanenin Oluşturulması

Sık kullanılacak araçlar sürekli yazılmaktansa onları bir kütüphane içine ekleyip her kullanılacak yerde aynı kütüphaneyi kullanmak zaman ve maliyet açısından fayda sağlayacaktır. İşte bu modülerliğin

getirmiş olduğu bir kolaylıktır. NetBeans ortamında kütüphane oluşturmak için File>New Project oradan da Java Class Library seçilir. İsim verildikten sonra oluştur dendiği vakit boş bir kütüphane projesi oluşacaktır. Bu adımda bir paket ekledikten sonra paketin içerisine bir sınıf eklenir. Bu sınıfın adını ve içeriğini aşağıdaki gibi yapınız.

```
public class PI {
   private final double hassasiyet;
   public PI(double denemeSayisi){
      hassasiyet = denemeSayisi;
   }
   public double Deger(){
      int basarilivurus=0;
      Random generator = new Random();
      for(double i=0;i<hassasiyet;i++){
         double x = generator.nextDouble();
         double y = generator.nextDouble();
         double uzunluk = Math.sqrt((Math.pow(x, 2)+Math.pow(y, 2)));
         if(uzunluk <= 1) basarilivurus++;
      }
      return 4*(basarilivurus/hassasiyet);
   }
}
```

Daha sonra kütüphaneye ters tıklayıp Clean and Build dediğimizde hata yoksa kütüphanemizin jar uzantılı dosyası oluşacaktır. Bu oluşan kullanılabilecek bir araç olup kendi başına çalıştırılamaz. Bu kütüphaneyi başka çalıştırılabilir bir uygulamada kullanmak için oluşturulan projenin Libraries kısmına gelinerek Add Jar/Folder demek koşuluyla oluşturmuş olduğumuz jar uzantılı kütüphanemizi ekleyebiliriz.

```
public static void main(String[] args) {
      // TODO code application logic here
      PI pi = new PI(1586516);
      System.out.print(pi.Deger());
   }
```

Hazırlayan
Dr. Öğr. Üyesi M. Fatih ADAK