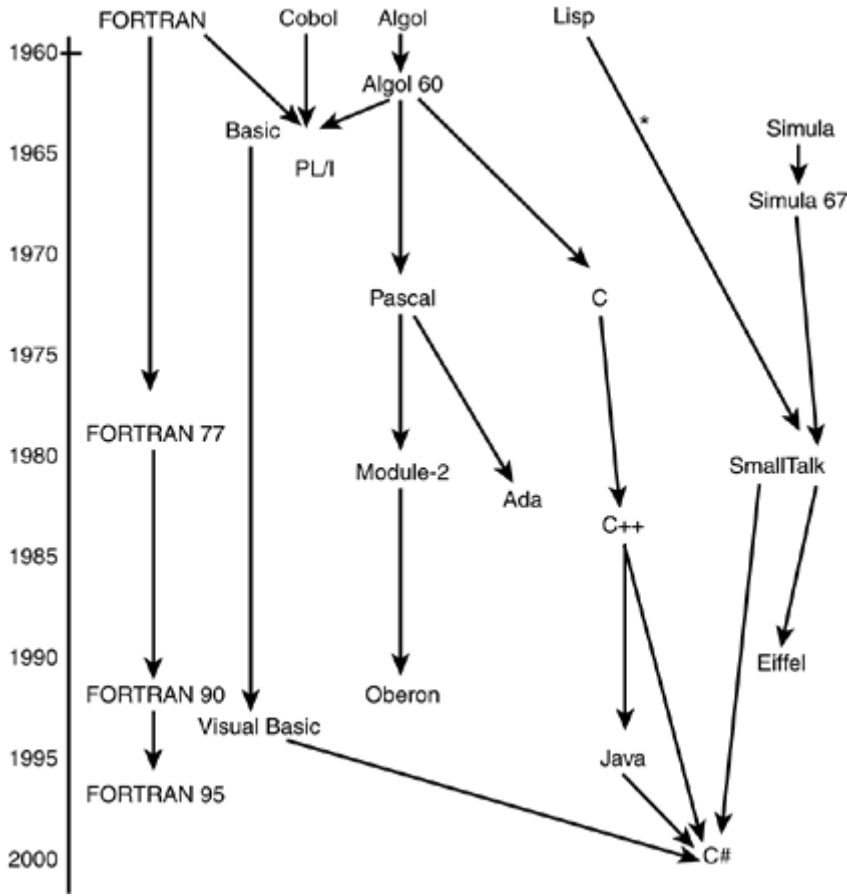


Programlama Dili Prensipleri

Lab Notları – 1

Programlama Dillerinin Tarihçesi:



C Programlama Dili

C dilinin ANSI standardı 1989’da onaylanmış fakat 1990 tarihinde yayınlanmıştır. C dili, hızlı ve düşük seviye özelliklere erişmek isteyen uygulamaların yazılması için popüler bir dildir. C programlama dili yapısal bir programlama dili olup nesne desteği yoktur. Aktif bellek yönetimini destekleyen C dilini kullanacak bir programcı bellek yönetimini çok iyi bilmelidir. Kaynak dosyaları genelde “.c” uzantısına sahiptir. Bu derste kodlar derlenirken bir GNU derleyicisi olan MinGW kullanılacaktır. MinGW kurulumunu yapmak için SABİS üzerinde paylaşılan dosya incelenebilir.

İlk Program

Girilen 4 basamaklı bir sayıyı basamaklarına ayırmak.

```
#include "stdio.h"
#include "locale.h"

int main()
{
    int sayi;
```

```

setlocale(LC_ALL,"Turkish"); // Türkçe karakter desteği için
do{
    printf("4 Basamaklı bir sayı girin:");
    scanf("%d",&sayi);
}while(sayi<1000 || sayi>10000); //4 basamaklı sayı kontrolü
short birler,onlar,yuzler,binler;
binler=sayi/1000;
yuzler=(sayi%1000)/100;
onlar=(sayi%100)/10;
birler=sayi%10;
printf("\nBinler:%d\n",binler);
printf("\nYüzler:%d\n",yuzler);
printf("\nOnlar:%d\n",onlar);
printf("\nBirler:%d\n",birler);
getchar();
return 0;
}

```

Nasıl Derlenir: Komut satırından kodun bulunduğu klasöre gelerek “gcc -o basamak basamak.cpp” girilip enter tuşuna basılırsa derleme gerçekleşip basamak.exe dosyası oluşacaktır.

Kod incelendiğinde yorum satırlarının // ifadesiyle başladığı görülecektir. Birden çok satırda yorum yapmak için /* yorumlar... */ şeklinde bir kullanım yapılmalıdır. Yorum satırları derleyiciler tarafından görülmez ve o satırlar atlanır. Bu satırlar programcılar için açıklama mahiyetindedirler.

karakteri ile başlayan ifadeler, ön işlemci komutlarıdır. Derleme işleminin hemen öncesinde çalıştırılırlar. Bir başka dosya veya kütüphane ekleme işlemleri bu şekilde yapılmaktadır.

{ } ifadeleri C dilinde kod bloklarının açılma ve kapanma ifadeleridir. Fonksiyonlar, yapı tanımlamaları döngüler, kontrol ifadelerinin hepsi birer kod bloğu olduğu için bu ifadeler ile başlayıp biterler. Fakat bu ifadeleri kullanmak için illaki bu bahsi geçen yapıların olmasına gerek yoktur. Aşağıdaki örnek kod C dilinde derlenip çalışacaktır.

```

#include "stdio.h"
int main(){
    int a=10;
    {
        printf("%d\n",a);
        {
            printf("Merhaba\n");
        }
    }
    return 0;
}

```

stdio.h kütüphanesinin eklenmesinin nedeni printf ve scanf gibi girdi ve çıktı fonksiyonlarının tanınması içindir. scanf girdi alma ifadesidir. printf ise çıktı verme için kullanılır.

Çalıştırılabilir bir C programı olabilmesi için bir main yani giriş fonksiyonuna ihtiyaç vardır. Bu derleyicinin kodun hangi satırından başlayacağını bilmesi içindir. main fonksiyonu tamamlandığında program kapanmış demektir. main fonksiyonun sonudaki return 0; ifadesi programın, işletim sistemine “her şey yolunda gitti ve işlem başarıyla tamamlandı” demesidir.

C dilinde çalıştırılacak tüm ifadeler ; ile biter. '\n' ifadesi ise bir alt satıra getirme karakteridir. C programlama dilinde kaynak kodda bırakılan boşluk miktarının bir önemi yoktur örneğin aşağıdaki program başarıyla çalışır. Sadece okunabilirliği düşüktür.

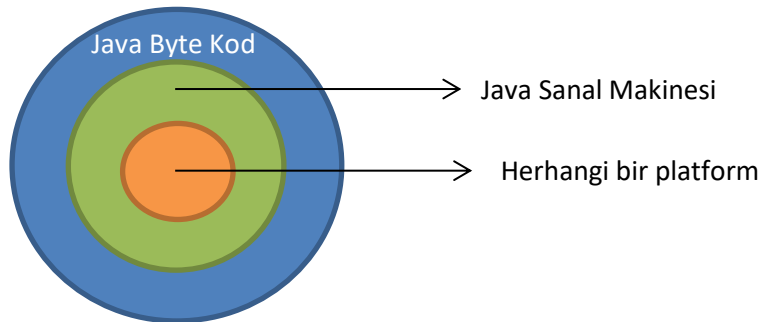
```
#include "stdio.h"
int main(){int i; for(i=0;
i<10;
i++)printf("%d\n",i); }
```

Java Programlama Dili

İlk başlarda **Oak** ismiyle tasarlanan Java programlama dili gömülü uygulamalar için kullanılıyordu. 1995 yılında Java ismini aldı. İnternet uygulamaları geliştirmek için yeniden tasarlandı. Bundan sonra Java gittikçe popülerleşti. Java'nın hızlı bir şekilde yükselişinin ve bu kadar popüler olmasının ardındaki en büyük etken verdiği sözde yatmaktaydı. "Kodu bir kere yaz ve istediğin yerde çalıştır." Java dilinin desteklediği özellikleri madde halinde yazacak olursak

- Tasarımı kolay
- Nesne Yönelimli
- Taşınabilen
- Yorumlanabilen
- Güvenli
- Yüksek Performanslı
- Çoklu-Thread destekleyen
- Dinamik

Günümüzde Java, sadece web uygulaması için değil bilgisayar uygulaması için de kullanılmaktadır. Yüksek seviyeli bir dilde yazılan programa kaynak program denir. Bilgisayar kaynak programı anlayamaz bunun için bir dönüştürücüye ihtiyaç vardır. Kaynak koddan makinenin anlayacağı assembly koda çevirenlere **derleyici** denir. Derleyiciler ekstra kütüphane ve nesne dosyalarını da **linker** yardımıyla derleme sürecine dahil ederler. Uygun derleyici yardımıyla bir program kodu herhangi bir makine koduna çevrilebilir. Fakat bir makine kodu sadece ilgili özel makinede çalışacaktır. İşte Java bu engeli ortadan kaldırmak için herhangi bir platformda çalışabilecek şekilde tasarlanmıştır. Bunun yaparken Java, kaynak kodu özel bir makine koduna çevirir. Bu makine koduna **Byte kod** (bytecode) denir. Bu Byte kod istediğiniz herhangi bir makinede Java sanal makinesi (Java Virtual machine) yardımıyla çalıştırılabilir.



Bir Java programı birçok yolla yazılabilir. Bir Java uygulaması, bir Applet veya bir Servlet olabilir. Uygulamalar herhangi bir bilgisayarda Java Sanal Makinesi yardımıyla çalıştırılabilen programlardır. Appletler web üzerinden çalışan özel uygulamalardır. Servlet'ler ise server üzerinden çalışan ve

dinamik web içeriği üreten programlardır. Bu ders kapsamında sadece bilgisayar üzerinde çalışan Java uygulamaları üzerinde durulacaktır.

Her Java programı en az bir sınıfa sahiptir. Sınıf metot ve verileri içeren bir yapıdır. Sınıflar “.java” uzantılı dosyalardır. Nesne dosyası oluşturulduğunda “.class” şeklinde bir dosya oluşur. Bu ders kapsamında Java derleyicisi olarak **NetBeans** kullanılacaktır. NetBeans kurulumu için aşağıdaki dosyadan yardım alabilirsiniz.

https://dosya.sakarya.edu.tr/Dokumanlar/2015/BSM208/352180945_netbeans_kurulumu.pdf

Örneğin aşağıdaki programı NetBeans’ta yazarsak. Program girilen ifadeyi ekrana yazmaktadır. Programdan da görüleceği üzere Java, C++ ile aynı altyapıdan geldiği için birçok yönden benzerlikler bulunmaktadır.

```
package ilkproje;

import java.util.Scanner;

/**
 *
 * @author M.Fatih
 */
public class IlkProje {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner scan = new Scanner(System.in);
        String girilen = scan.nextLine();
        System.out.println(girilen);
    }
}
```

Yorum açısından C ve C++ ile aynıdır. `/** */` bu şekilde bir yorum tarzı dokümantasyon için kullanılır. İlk satırda bulunan `package`, paket tanımlaması için kullanılır. Amacı isim benzerliğinin ve bundan doğacak problemlerin önüne geçmektir. Paket ismi klasör ismi ile aynı olmalıdır. Örneğin yukarıdaki `IlkProje` sınıfı `ilkProje` klasörünün içerisinde yer almaktadır. Farklı bir paket ismi de olabilirdi. Buradaki önemli nokta paket ismi ile klasör isminin aynı olmasıdır. Java’da kütüphanelerin, sınıfların veya farklı dosyaların yazılan projeye eklenmesi `import` kelimesi ile olmaktadır. Yukarıdaki kodda girdi almak için kullanılan `Scanner` sınıfı `import` kelimesi ile projeye eklenmiştir.

`import java.util.*;` * karakteri de kullanılabilir. * karakteri paketin hepsini dahil eder. `util` paketindeki her şey dahil edilecektir. Eğer `import` yazılmadan kullanılmak isteniyorsa bu durumda her kullanışta tam paket yolu yazılmalıdır.

`java.util.Scanner scan = new java.util.Scanner(System.in);` gibi

C ve Java’da ayrılmış kelimeler (her ikisi içinde geçerlidir)

break case catch char struct const continue default do double else float for goto if int long malloc return short switch throw try void volatile while

Bu kelimelerden const ve goto artık Java için tanımsız olup kullanılmamaktadır.

Aşağıdaki kelimeler ise sadece Java için özeldir.

abstract boolean byvalue cast extends final finally future generic implements import inner instanceof interface native null operator outer package rest super synchronized this throws transient var

Yukarıda kırmızı ile yazılanlar şuanda Java’da tanımsız olup kullanılmamaktadır.

Java’da en basit program yazmak için bile sınıf tanımlamak bir zorunluluktur. Fakat Java’da main metodunun başında static ifadesi bulunmaktadır. Bu ifadelerin daha detaylı bir şekilde anlatımı ileriki bölümlerde açıklanacaktır. static konmasının sebebi main metodu nesne oluşumu başlamadan önce çalışmaya başlıyor. Eğer static olmasaydı nesnesi oluşturulup metodun çağrılması gerekecekti.

```
/* Dene.java */
public class Dene {
    public void F(){
        System.out.println("Cagildi...");
    }
}

Dene d = new Dene();
d.F();
```

Eğer F metodu static tanımlansa idi aşağıda görüldüğü gibi Dene sınıfının nesnesi oluşturulmadan çağrılabilirdi.

```
/* Dene.java */
public class Dene {
    public static void F(){
        System.out.println("Cagildi...");
    }
}

Dene.F();
```

Komut Satırı Parametreleri

Komut satırı parametreleri, daha program çalışmaya başlamadan önce programa belli parametreleri girmeyi sağlar. Bu C ve Java’da main metodunun parametreleri ile sağlanır. Örneğin aşağıdaki iki kod sırasıyla C ve Java’da komut satırından aldığı parametreyi ekrana yazar. C’de 1. İndekstekini almamızın sebebi 0. İndekste programın kendisini tutmaktadır. Bunun dışında C’de argc isminde bir başka parametrede bulunmaktadır. Bu argc, komut satırından kaç adet parametre girildiğini gösterir. Java’da Netbeans ortamında komut satırı parametresi girebilmek için proje adına ters tıklayıp “**Properties**” kısmına geliriz. Buradan da Run alt alanına gelindikten sonra “**Arguments**” yazan kısım komut satırı parametrelerini ifade eden kısımdır. Birden çok parametre girmek için parametreler arası boşluk bırakılmalıdır.

```
// Java
public static void main(String[] args) {
```

| |
|--|
| <pre>// TODO code application logic here System.out.println(args[0]); }</pre> |
| <pre>// C Dili #include "stdio.h" int main(int argc, char *argv[]){ printf("%s\n", argv[1]); return 0; }</pre> |

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

Programlama Dillerinin Prensipleri

Lab Notları – 2

Veri Türleri

Karakter Tipi (char): C/C++ dillerinde karakter tek bir byte ile ifade edilir. Bir byte 8 bit olduğu için, en fazla 256 karakter ifade edilebilir. Bunlar ASCII kodu olarak ta bilinirler. Fakat Java’da Unicode kullanılmaktadır. Bir karakter için 2 byte ayrılır. Bu da 65536 karakter yazılabilir anlamına gelir. Böylelikle latin harflerinin dışında birçok farklı karakter yapıları da sığdırılabilmektedir.

C ve Java’da \ karateri çıkış karakteri olarak kullanılır. Özel bir karakterdir.

\n Yeni satır
\b Bir karakter geri
\t Tab
\' Tek karakter koymak için
\" Çift karakter koymak için

```
#include "stdio.h"
int main(){
    char c='\b';
    printf("Merhaba%c%c",c,c);
    getch();
    return 0;
}
```

Yukarıdaki kod bloğunda c içerisinde bir karakter geri ifadesi tutulduğu için ekrana Merhaba yazdığında imleç b’nin üzerinde yanıp sönecektir.

Aşağıdaki özel karakterler C/C++’ta bulunmasına rağmen Java’da yoktur.

\a ses çıkarır
\? Soru işareti
\v Dikey tab

Java programlama dilinde karakteri ifade etmek için farklı özel bir yolu vardır. Karakteri hexadecimal değerini yazdırarak ta ekrana çıkartabilirsiniz.

```
public class IlkProje {
    public static void main(String[] args) {
        char a='\u0391';
        System.out.println(a); // Ekrana A harfini yazar.
    }
}
```

Boolean Tipi: Java’da boolean olarak tanımlanan doğru ve yanlış veri türü bellekte 1 bit yer kaplamaktadır, standart C programlama dilinde ise bu türe yer verilmemiştir. C dilinin doğru ve yanlış durumlara bakış açısı biraz farklıdır. Sıfır değeri yanlış kabul edilip bu değer dışındaki bütün değerler doğru olarak kabul edilir. Örneğin aşağıdaki kod parçasında ekrana Sakarya yazacaktır.

```
#include "stdio.h"
int main(){
    if(200) printf("Sakarya");
    else printf("Ankara");
    return 0;
}
```

Yukarıdaki ifade biraz daha iyileştirilirse aşağıdaki gibi tanımlanır ve Java benzeri bir doğru yanlış veri türü elde edilmiş olur. Yapılan şey false ve true iseminde iki kelimenin 0 ve 1 ile ilişkilendirilmesi ve bool ismi ile ifade edilmesidir.

| | |
|---|---|
| <pre>#include "stdio.h" typedef enum {false, true} bool; int main(){ bool x=true; if(x) printf("Sakarya"); else printf("Ankara"); return 0; }</pre> | <pre>#include "stdio.h" typedef enum {false, true} bool; int main(){ bool x=true; if(x == true) printf("Sakarya"); else printf("Ankara"); return 0; }</pre> |
|---|---|

Java'da int, float long, double gibi bütün ilkel türlerin boyutları her platformda **sabittir**. Bu taşınabilirliğin getirmiş olduğu bir zorunluluktur. Bundan dolayıdır ki Java'da sizeof operatörü yoktur. Fakat C dilinde durum bu şekilde değildir. Mimariden mimariye ilkel türlerin kaplamış oldukları alanda farklılıklar olabilir. Aşağıdaki kod çalıştırıldığında ekrana 4 yazacaktır. Bu int ilkel türünün bellekte 4 byte kapladığı anlamına gelir. X değişkenine atanan sayının büyüklüğü ile bellekte kapladığı yer arasında bir bağlantı yoktur. Örneğin kodun ikinci kısmında ekrana tekrar 4 byte yazacaktır. Eğer 4 byte'a sığmayacak bir sayı kullanılmak isteniyorsa örneğin double türü düşünülebilir. Aşağıda sizeof'un neden bir fonksiyon değil de operatör olarak ifade edildiği sorulursa aşağıdaki yazılış şekliyle anlaşılabilir.

| | |
|---|--|
| <pre>#include "stdio.h" int main(){ int x=100; printf("%d",sizeof x); return 0; }</pre> | <pre>#include "stdio.h" int main(){ int x=1000000000; printf("%d",sizeof x); return 0; }</pre> |
|---|--|

C dilinde ilkel türler kategori olarak ikiye ayrılırlar, kayan noktalı (ondalık) ve tamsayı olan türler. char, short, int ve long tamsayı türlerine girer. float, double ve long double ise kayan noktalı türlere girer.

Double ile float arasındaki farka bakıldığında Java ve C dilleri için söylenebilecek şey, double türünün, float türüne göre ondalık kısmının daha fazla olmasıdır. Aşağıdaki örnek kod C dilinde yazılmış ve double ile float ayrı ayrı kullanılmıştır. Oluşan ekran çıktılarında double'ın daha doğru bir ondalık kısım gösterdiği görülmüştür. Aynı durum Java için de geçerlidir.

| | |
|---|--|
| <pre>#include "stdio.h" int main(){ float x=10; double a=10; float y=3; double b=3; float z = x/y; double c = a/b; printf("float: %.10f\n",z); printf("double: %.10f\n",c); }</pre> | <pre>public static void main(String[] args) { float x=10; double a=10; float y=3; double b=3; float z = x/y; double c = a/b; System.out.println("float:"+z); System.out.println("double:"+c); }</pre> |
|---|--|

| | |
|--|--|
| <pre> return 0; } </pre> | <pre> } </pre> |
| Ekran Çıktısı: float: 3.3333332539 double: 3.333333333 | Ekran Çıktısı: float:3.3333333 double:3.3333333333333335 |

Yine aynı sebeplerden aşağıdaki karşılaştırma hem Java hem de C dilinde false değerini döndürecektir.

| | |
|--|--|
| <pre> int main(){ float x=0.1; double y=0.1; if(x == y) printf("x ve y esit"); else printf("Esit degil"); return 0; } // False değerini döndürür. </pre> | <pre> public static void main(String[] args) { float x=0.1f; double y=0.1; if(x == y) System.out.print("x ve y eşit."); else System.out.print("Eşit değil."); } // Javada ondalık sayılar varsayılan olarak // double olduğu için float olarak tanımlak // sonuna f getirmekle mümkündür. </pre> |
|--|--|

Tür dönüşümlere bakıldığında, C ve Java'da da küçük veri türünden büyük veri türüne dönüştürüldüğünde bir sıkıntı oluşmamaktadır.

| | |
|---|--|
| <pre> public static void main(String[] args) { int x=100; double a=x; System.out.println(a); } </pre> | |
| <pre> int main(){ int x=100; double a=x; printf("%f",a); return 0; } </pre> | |

Sıkıntı **büyük veri türünden küçüğüne** dönüştürüldüğünde ortaya çıkmaktadır. C dili esnekliği gereği herhangi bir derlenme hatası vermez. Fakat dönüştürülen değer boyutu daha küçük olan veri türüne sığmayacaksa veri kaybı olur. Örneğin aşağıdaki C kodunda ondalık değer tamsayıya dönüştürülmüş ve ondalık kısmı kaybolmuştur.

| |
|--|
| <pre> int main(){ double x=100.35; int a=x; printf("%d",a); return 0; } </pre> |
|--|

Fakat aynı dönüşüme Java izin vermez ve derlenme anında hata verir. Hatadan kurtulmak için atamanın başına (int) getirilmelidir. Bu Java derleyicisine veri kaybının farkındayım mesajını vermektir. Fakat yine veri kaybının önüne geçilemez.

| |
|---|
| <pre> public static void main(String[] args) { double x=100.45; int a=(int)x; System.out.println(a); } </pre> |
|---|

Java’da türler küçük harf ile başlıyorsa ilkel tür büyük harf ile başlıyorsa o ilkel türün sınıfı olduğunu gösterir. Örnek: Double , double gibi. Ek özellikler kullanılmak isteniyorsa sınıf olanı kullanılmalıdır.

Sabitler

Bazen program yazılırken bazı değerlerin programın sonuna kadar sabit kalması istenebilir. Örneğin pi sayısı veya kat sayılar gibi. Aşağıdaki kod incelendiğinde 9.81’in aslında orada bir sabit olduğu ve değişmemesi gerektiği görülecektir. Fakat kodu analiz eden bir başka programcı 9.81’in belki de sabit olduğunu anlayamayacaktır.

```
public static void main(String[] args) {  
    double kuvvet,kutle=78;  
    kuvvet = kutle * 9.81;  
    System.out.println(kuvvet);  
}
```

Bunun yerine aşağıdaki gibi kullanılması daha açıklayıcı olacaktır.

```
public static void main(String[] args) {  
    final double yercekimi =9.81;  
    double kuvvet,kutle=78;  
    kuvvet = kutle * yercekimi;  
    System.out.println(kuvvet);  
}
```

Yukarıdaki kodda görüldüğü gibi final bir ifadeyi sabit yapmak için kullanılır. C dilinde de bu özellikler geçerli olup sabit tanımlamak için const ifadesi kullanılır.

```
int main(){  
    const double pi=3.14;  
    double yariCap=5.2;  
    printf("Cevre=%.2f",2*pi*yariCap*yariCap);  
    return 0;  
}
```

Fakat Java ve C dili arasında sabit tanımlamada önemli bir fark bulunmaktadır. Java’da sabite vereceğiniz değer kullanılmadan önce herhangi bir satır olabilir. Fakat C dilinde sabitin tanımlandığı yerde değerini alması gerekmektedir.

```
public static void main(String[] args) {  
    final double yercekimi;  
    double kuvvet,kutle=78;  
    yercekimi=9.81; // C dilinde bu kullanıma izin verilmez.  
    kuvvet = kutle * yercekimi;  
    System.out.println(kuvvet);  
}
```

Hazırlayan
Arş. Gör.Dr. M. Fatih ADAK

Programlama Dili Prensipleri

Lab Notları – 3

Veri Türleri - 2

Diziler

Homojen verilerin bir araya gelerek oluşturdukları yapı. Bir dizi içerisinde aynı tür veri bulunur. Dizi indeksi sıfırdan başladığı için son indeks elemansayısı – 1 olarak ifade edilir. C ve Java’da dizi tanımlamaları birbirine yakındır. Aşağıdaki örnek C dilinde çalışırken Java’da ilklenden kullanılmaya çalışılıyor hatası verecektir. Burada aslında ileride anlatılacak olan bellek ile alakalı bir durum söz konusudur. Java’da diziler tanımlandıkları yerde değerleri verilmeli ya da heap bellek bölgesinde oluşturulmalıdırlar.

```
#include "stdio.h"
int main(){
    int x[5];
    x[0]=100;
    printf("%d",x[0]);
    return 0;
}
```

```
int x[5];           // Derlenme zamanı hatası verir.
x[0]=100;
System.out.println(x[0]);
```

Java için doğru tanımlama aşağıdaki iki şekilde olabilir.

```
int[] x={100,200,300};
System.out.println(x[0]);
```

```
int[] x=new int[3];
x[0]=100;
System.out.println(x[0]);
```

İki boyutlu dizilerde tanımlama yukarıdakine benzer koşullarda aynıdır. Burada dikkat edilmesi gereken dizilerin arka planda aslında bir gösterici şeklinde tutulduklarıdır. Dolayısıyla aşağıdaki iki boyutlu dizi tanımlamasında ekrana adres yazacaktır.

```
int x[3][3];
x[0][0]=100;
printf("%d",x[0]);
```

```
int [][] x = new int[3][3];
x[0][0]=100;
System.out.println(x[0]);
```

Dizilerin bellekte tutulma şekilleri bir göstericinin bellekte tutulma şekli ile aynıdır. Yapılan iş sadece ilk elemanın adresini tutmaktır. Derleyici dizinin ilk elemanın adresini tutmakla yetineceği için diziler tanımlandıkları yerde boyutları belirtilmelidir ki derleyici adresi nereye kadar arttırabileceğini bilsin. Siz sayılar[3]’teki elemanı getir dediğinizde derleyici arka tarafta aslında *(sayılar+3) adresindeki değeri getir demektedir.

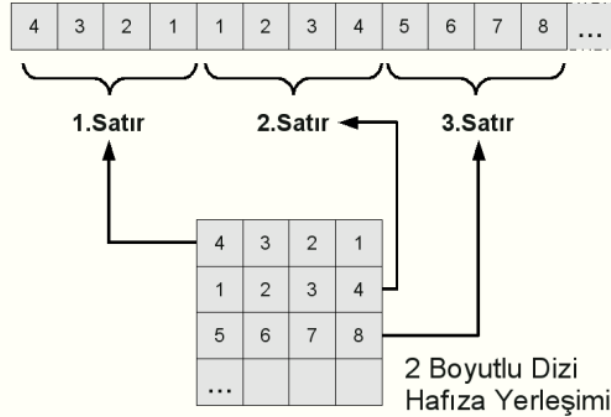
Programda sayıların tutulduğu adresleri ekrana yazmaya kalkarsanız. Adreslerin ardışık olduğunu göreceksiniz.

2686696

2686700

2686704

Adreslere bakıldığında 4 farkla ilerlediği görülür. $96+00=04$ hexadecimal olarak. Bunun nedeni int bellekte 4 byte olarak tutulduğundan kaynaklanmaktadır (C Dili).



Set Türü: Set veri türünü programlama dilleri arasında sadece Modula-2 ve Pascal destekler. Diğer programlama dillerinin özellikle Java gibi dillerin desteklememesinin en büyük nedeni set bellekte Word boyutuna göre yer ayırır. Buda demek oluyor ki bir makinede yazmış olduğunuz program farklı bir mimariye sahip diğer bir makinede Word boyutu farklı olduğu için çalışmayabilecektir. Buda taşınabilirliği engeller.

Pointers (Göstericiler)

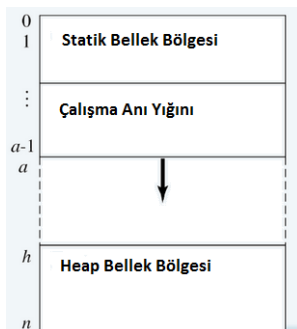
Göstericilerin içinde tuttukları değer adres değerleridir. Göstericilerin geliştirilme amacı dolaylı (indirect) adreslemenin gücünden faydalanmak (daha çok makineye yakın dillerde kullanılır.) ve dinamik bellek yönetimini sağlamak içindir. Bu bellek bölgesine heap bellek bölgesi adı verilir. Java gibi üst düzey dillerde belleğe doğrudan erişime izin verilmez. Fakat C dili ile yapılan pointer işlemleri daha kısıtlanmış hali Java'da yapılır. Aslında Java'da kullanılan nesnelere erişim şekilleri referanslar yardımıyla yapılmaktadır. Ama Java'da C dilindeki gibi * ile erişim yapılmamakta heap bellek bölgesini gösteren zaten bir referans olmaktadır. Aşağıdaki örneği inceleyelim. Her iki kod bloğunda da aslında p ve r birer referanstır.

| | |
|---|---------------|
| <pre>#include "stdio.h" int main(){ int x=100,y=50; int *p = &x; int *r = &y; int *tmp = p; p=r; r=tmp; printf("p:%d\n",*p); printf("r:%d\n",*r); return 0; }</pre> | C Dili Örneği |
| <pre>public class Sayi { public int deger; public Sayi(int dgr){ deger=dgr; } }</pre> | Java Örneği |

| | |
|--|--|
| <pre> } } public class IlkProje { /** * @param args the command line arguments */ public static void main(String[] args) { Sayi p = new Sayi(100); Sayi r = new Sayi(50); Sayi tmp = p; p=r; r=tmp; System.out.println("p:"+p.deger); System.out.println("r:"+r.deger); } } </pre> | |
|--|--|

Çalışan herhangi bir programdaki değişkenler, sınıflar, metotlar mutlaka bellekte bir yerde tutulurlar. Bellekte tutuldukları yerler bakımından 3 farklı bölge bulunmaktadır.

- Statik Bellek Bölgesi
- Çalışma Anı Yığını
- Heap Bellek Bölgesi



Statik Bellek Bölgesi

Bu bölgede yer alacak değişkenler hangileri olduğu, daha program başlamadan bellidir. Bu bölgede Global değişkenler, sabitler, static olarak tanımlanmış lokal değişkenler tutulurlar. Statik bellek bölgesinde tutulan değişkenler program çalıştığı sürece var olurlar, program sonlandığında bellekten silinirler.

```

#include "stdio.h"
int kontrol;
const float pi=3.14;
int Arttir(){
    static int sayim = 0;
    sayim++;
    return sayim;
}
int main(){
    printf("%d\n",Arttir());
    printf("%d\n",Arttir());
}

```

```
printf("%d\n",Arttir());  
}
```

Yukarıdaki C kod örneğinde kontrol değişkeni global değişkendir. pi sayısı const ifadesi olduğu için sabittir ve Arttir metodunun içerisindeki sayim değişkeni de başında static olduğu için statik lokal değişkendir. Bu ismi anılan 3 değişkende statik bellek bölgesinde tutulur. Statik bellek bölgesinde tutulduğu için program sonlanıncaya kadar bellekte tutulurlar. Bundan dolayı Arttir metodu her çağrıldığında sayim değişkeni sıfırdan başlamak yerine kalmış olduğu değerden devam edecektir. Java’da metot içerisinde static kullanımına izin yoktur. Sadece sınıf içerisindeki elemanlar static olarak tanımlanabilir. Bununda anlamı bu eleman sınıftan türetilen bütün nesneler için ortak ve aynıdır. Aşağıdaki örnek incelendiğinde ekrana her zaman aynı sayıyı yazacaktır. Bunun da nedeni basittir, static olarak tanımlanmış Sayi sınıfının deger elemanı tüm nesneler için ortak olacak ve üstünde yapılmış en son değişikliği koruyacaktır.

```
public class Sayi {  
    public static int deger;  
    public Sayi(int dgr){  
        deger=dgr;  
    }  
}  
public static void main(String[] args) {  
    // TODO code application logic here  
    Sayi p = new Sayi(100);  
    Sayi r = new Sayi(50);  
    Sayi tmp = p;  
    p=r;  
    r=tmp;  
    System.out.println("p:"+p.deger);  
    System.out.println("r:"+r.deger);  
}
```

Global değişkenler statik bellek bölgesinde tutuldukları için program sonlanıncaya kadar bellekte tutulacaktır ve programın herhangi bir satırından bu değişkenlere erişilebilecektir. İşte bu yüzden global değişkenlerin kullanımı (değişip değişmediklerinin kontrolü zor olduğu için) tavsiye edilmemektedir.

Çalışma Anı Yığını

En aktif bellek bölgesidir denilebilir. İsmi de oradan aldığı bu bellek bölgesi bir yığın (stack) şeklindedir ve bu yapıda çalışır. Bu yapıya ilk giren en son çıkar. Bir program çalıştığı sürece genişleyip daralan bitişik bir yapıya sahiptir. Bu bellek bölgesinde fonksiyon ve metot çağrımları ve bu fonksiyon ve metotların barındırdığı lokal değişkenler bulunur. Bir fonksiyon veya metot çağrıldığında bu fonksiyon veya metoda ait parametreler değişkenler ve dönüş değerleri bu bellek bölgesinde tutulur. Çalışma anı yığın bölgesi genişlemiş olur. Fonksiyon veya metot çağrılan yere döndüğünde bu fonksiyon veya metodun çalışma anı yığnında ayırmış olduğu yer geri döndürülür. Dolayısıyla geri döndürülen bu değişkenlere çağrı bittikten sonra erişim olamayacaktır.

```
#include "stdio.h"  
int DegerArttir(){  
    static int sayac=0; //Statik bellek bölgesinde  
    return ++sayac;  
}
```

```

int topla(int a,int b){
    int sonuc = a+b; // Çalışma anı yığnında
    return sonuc;
}
int main(){
    printf("%d\n",DegerArttir());
    printf("%d\n",DegerArttir());
    printf("%d\n",topla(21,10));
    printf("%d\n",topla(5,7));
    return 0;
}

```

Yukarıdaki kod örneğine bakıldığında, iki metot ve bir main metodu yer almaktadır. Main metodunda iki kere DegerArttir metodu çağrılmış ve daha sonra topla metodu çağrılmıştır. DegerArttir metodunun içerisindeki sayaç değişkeni statik lokal değişken olduğu için statik bellek bölgesinde diğer bütün değişkenler, çalışma anı yığnında oluşturulur. Topla metodunun çağrımı bittikten sonra, çalışma anı yığnında oluşturulmuş olan a, b ve sonuc değişkenleri bellekten yok edilirler.

Heap Bellek Bölgesi

Bu bellek bölgesi C ve C++ gibi programlama dillerinde dikkat edilmesi gereken çok önemli bir bölgedir. Çünkü C ve C++ gibi dillerde bu bölgenin kontrolü programcıya bırakılmıştır. Bu da demek oluyor ki eğer bu bölgenin kontrolü iyi sağlanmaz ise bellek taşması ya da yanlış değerlere erişim gibi problemler ile karşı karşıya kalınabilir. Bu bölgeye duyulan ihtiyacın nedeni, dinamik oluşturulan yapıların boyutları değişken olacak ve çalışma anında belirlenecektir. Dolayısıyla bu yapılar heap bellek bölgesinde tutulmalı, bu yapılara ve değişkenlere göstericiler yardımıyla erişilmelidir. Bu bellek bölgesinde tutulan bir değer in adı yoktur yani anonimdir ve ancak değer in bulunduğu adresi gösterecek bir gösterici yardımıyla erişilebilir.

Heap bellek bölgesinde C programlama dilinde bir yer ayırmak için malloc Java'da ise new operatörü kullanılır. C dilinde malloc kullanmadan tanımlanan göstericiler heap bellek bölgesinden yer ayıramazlar. Aşağıdaki C ile yazılmış örneği inceleyelim.

```

#include "stdio.h"
#include "stdlib.h"
int main(){
    int *yas = malloc(sizeof(int)); // Heap bellek bölgesi
    *yas = 30;
    printf("%d\n", *yas);
    int *p; // adresi yok
    free(yas);
    return 0;
}

```

malloc metodu kullanıldığında "stdlib.h" kütüphanesi programa eklenmelidir. Heap bellek bölgesinde ayrılan yer C dilinde, işi bittiğinde belleğe geri verilmelidir. Bu bölgenin kontrolü programcıda olduğu için eğer geri döndürülmez ise çöp dediğimiz durum oluşur. Hatırlanırsa bu bölgedeki adreslere çalışma anı yığnındaki göstericiler yardımıyla erişiliyordu. Dolayısıyla çalışma anı yığnındaki gösterici kaybedilmeden önce heap bellek bölgesinde ayrılan yer geri döndürülmelidir. Yoksa o bölge bilgisayar kapanıncaya kadar kullanılamaz duruma gelir. Java'da ise bu durumda yani ayrılan yer belleğe geri verilmediği durumda yine çöp oluşur fakat Java'da çöp toplayıcı mekanizması vardır. Dolayısıyla belli aralıklarla çöp toplayıcılar devreye girerek göstericisi olmayan bellek bölgesini geri döndürürler. C

dilinde geri döndürme işi free metodu ile yapılır. Fakat Java’da buna benzer bir yapı yoktur. İllaki Java’da geri döndürülmek isteniyorsa bunun en güzel yolu null’a eşitlemektir. Böylelikle Java Sanal makinesinin çöp toplayıcısı çalıştığında geri döndürülecektir.

```
Sayi s = new Sayi(100);    // Java
s=null;
```

Burada tekrar geri dönüp bir deneme yapılırsa, hatırlarsanız Sayi sınıfının değer elemanı static olarak tanımlanmıştı. Yukarıdaki kod bloğunda s, Sayi nesnesini gösteren bir göstericidir. null'a eşitlenmiş ve geri döndürülmüştür. Fakat bir alt satırına değer yazdırılmaya çalışılırsa hata vermediğini ve 100’ü ekrana yazdığı görülecektir.

```
Sayi s = new Sayi(100);
s=null;
System.out.println(s.deger); // ekrana 100 yazar
```

void Göstericisi

C dilinde, türü olmayan bir göstericidir. Dolayısıyla yeri geldiğinde bir tamsayıyı gösterebileceği gibi yeri geldiğinde bir ondalık sayıyı da gösterebilir. Sadece göstericinin gösterdiği yer kullanılacağı zaman, derleyicinin o anki hangi tür olduğu bilmesi açısından dönüştürme işlemi uygulanmalıdır. Bunun örneği aşağıdaki kod parçasında görülebilir.

```
#include "stdio.h"
#include "stdlib.h"
int main(){
    int x=100;
    float a=12.5;
    void* obj;
    obj=&x;
    printf("%d\n",*(int*)(obj));
    obj=&a;
    printf("%.2f\n",*(float*)(obj));
    return 0;
}
```

Bu şekilde bir nevi object türüne benzetilebilir. Fakat Java’da buna gerek yoktur. Çünkü Java’da Object türü bulunmaktadır. Aşağıdaki örneği inceleyelim.

```
Object x=100;
System.out.println(x);
x="Sakarya";
System.out.println(x);
x=28.12;
System.out.println(x);
```

İşlemler

Java ve C dili dört işlemi desteklerler ve bunlar için özel operatörleri vardır. Bunlar dışında arttırma ve azaltma gibi operatörleri de desteklerler. İşlemlerde dikkat edilmesi gereken işleme giren operandlardan büyük tür, sonucunda türüdür. Örneğin aşağıdaki C kodunda ekrana 3 yazacaktır.

Sebebi işleme giren x ve y değişkenleri tam sayıdır ve sonucunda tam sayı olması gerekir. Bundan dolayı 3.5 **yazmamıştır**.

```
int main(){
    int x=7,y=2;
    float z=x/y;
    printf("%f\n",z);
    return 0;
}
```

Java programlama dilinde de durum aynıdır. Sonucun bir double'a atanması da durumu değiştirmeyecektir. Aşağıdaki Java programı çalıştırıldığında ekrana 3.0 yazacaktır.

```
int x=7,y=2;
double z=x/y;
System.out.println(z);
```

Doğru sonuç elde edilmesi için bir sayının daha büyük türe dönüştürülmesi gerekir. Örneğin aşağıdaki gibi yazılırsa sonuç 3.5 çıkacaktır.

```
int x=7;
double y=2;
double z=x/y;
System.out.println(z);
```

Programlama dillerinde işlem öncelikleri vardır. İlk önceliği parantezler belirler.

Öncelik Sırası:

+, - Sayıların işaretleri

++, -- Arttırma, azaltma

., * işaretçiler (pointer)

*, / , % Çarpma, bölme, modüler

+, - Toplama, çıkarma

-= , += , %= , /= , *= Bileşik atama işlemleri

= Atama işlemi

Örneğin aşağıdaki kod ekrana 50 yazar.

```
System.out.println(5+3*15);
```

Örneğin aşağıdaki C kodu ekrana 24 yazar çünkü arttırmanın önceliği çarpmadan daha yüksektir.

```
int main(){
    int x=1,y;
    y=++x*12;
    printf("%d\n",y);
    return 0;
}
```

Ama aynı kod aşağıdaki gibi yazıldığında ekrana 12 yazar sebebi arttırma işleminin daha sonra yapılmasıdır. Arttırmanın önceliği daha yüksek olabilir fakat ++ işareti x değişkeninden daha sonra geldiği için x önce işlemegirer daha sonra x'in değeri arttırılır. Fakat sonuç bundan etkilenmeyecektir ve 12 olacaktır.

```
int x=1,y;  
y=x++*12;  
printf("%d\n",y);
```

$X=X+10$; ifadesinde normalde soldan sağa ve yukarıdan aşağıya işleme sokulur. Fakat X'in değerinin hesaplanması için sağ tarafa ihtiyaç vardır. Dolayısıyla X'in eski değeri ile 10 toplanıp, X yeni değerini alacaktır. Tabi ki bu ifadenin kısaltmasını da programlama dilleri desteklemektedir.

$X += 10$; Bütün operatörlerde bu geçerlidir.

Atamaların her zaman sol tarafı değer alan kısım olmalıdır. Yani aşağıdaki tanımlama anlamsız ve geçersizdir. Atama sonrasında sol taraf değer kazanır.

$100 = x$;

Hazırlayan: Arş. Gör. M. Fatih ADAK

Programlama Dili Prensipleri

Lab Notları – 4

1. Karar Yapıları

IF Yapıları

Karar yapıları olarak C/C++ ile Java programlama dilleri birbirine yakın ifadeler içerir. Bir programın akışı yukarıdan aşağı doğru ilerler. Bu ilerleyişte bazı satırların bazı koşullarda çalıştırılması istenebilir. Bu durumda kontrol blokları kullanılmalıdır. Karar yapıları, if-if else if else , ternary operator ve switch case şeklinde kullanılabilir. Aşağıda C dilinde basit bir if karar yapısı görülmektedir.

```
#include "stdio.h"
int main(){
    int x;
    printf("Bir sayi girin:");
    scanf("%d",&x);
    if(x % 2 == 0) printf("Girilen sayi cifttir.\n");
    return 0;
}
```

Aynı programı çok fazla ifadeyi değiştirmeden Java'da aşağıdaki gibi yazabiliriz.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int x;
    System.out.print("Bir sayi girin:");
    x = in.nextInt();
    if(x % 2 == 0) System.out.println("Girilen sayi çifttir.");
}
```

Peki birden çok kontrol yapılması gerekiyorsa ne yapılmalıdır? Yapılacak işlem if sayılarını çoğaltmak olabilir. Aynı ayrı if blokları kullanılabileceği gibi içi içe de if blokları kullanılabilir. Burada dikkat edilmesi gereken blokları { } parantezleri ile ayırmaktır. Fakat if içerisinde çalıştırılacak bir ifade varsa parantezlere gerek olmaz.

```
#include "stdio.h"
int main(){
    int x;
    printf("Bir sayi girin:");
    scanf("%d",&x);
    if(x % 2 == 0)
        if(x < 100)
            if(x > 10)
                printf("Girilen sayi 10'dan buyuk 100'den kucuk bir cift sayidir.\n");
    return 0;
}
```

Yukarıdaki aynı durum Java'da da geçerlidir. Yukarıdaki kodu Java'da aşağıdaki gibi genişletirsek yine değişen bir şey olmayacak ve blokları ayıran parantezlere gerek kalmayacaktır. Ama kullanılması da bir hataya sebebiyet vermez. Bu kural aynı şekilde C dili için de geçerlidir.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int x;
    System.out.print("Bir sayı girin:");
    x = in.nextInt();
    if(x % 2 == 0)
        if(x < 100)
            if(x > 10)
                System.out.println("Girilen sayı 10'dan büyük 100'den küçük ve çift bir sayıdır.");
            else if(x > 5)
                System.out.println("Girilen sayı 5'ten büyük 100'den küçük ve çift bir sayıdır.");
            else
                System.out.println("Girilen sayı 5'ten küçük ve çift bir sayıdır.");
        }
    }
}

```

Switch-case Yapısı

Kontrol edilecek değerler kesin olarak belli ise switch case yapısı kullanmak daha uygundur. Belirli olmasından kasıt örneğin kullanıcının gireceği bir x değerinin belli sayılardan büyük, belli sayılardan küçük kontrolü if yapısına uygun iken girilecek değer sadece 2 veya 3 yani belli sayıda değer alabiliyorsa switch case yapısı daha uygundur. switch case yapısında switch ifadesinin içindeki değişkenin türü ne ise case ifadeleri o türde kontrol edilmelidir. Örneğin aşağıdaki Java örneğinde kullanıcıdan bir ülke adı girilmesi istenmiş ve ülke adına göre ya yurt içi ya yavru vatan ya da yurt dışı ekrana yazdırılmıştır. Dikkat edileceği üzere kullanıcıdan string türde değer alındığı için case ifadeleri string'leri kontrol etmektedir.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    String ulke;
    System.out.print("Bir ülke girin:");
    ulke = in.nextLine();
    switch(ulke)
    {
        case "Türkiye":
            System.out.print("Yurt İçi");
            break;
        case "Kıbrıs":
            System.out.print("Yavru Vatan");
            break;
        default:
            System.out.print("Yurt Dışı");
            break;
    }
}

```

Yukarıdaki default ifadesi switch case yapısının önemli bir unsurudur. default, eğer girilen değer hiçbir case ifadesine uymuyorsa çalışacak olan bloktur. Yazılması zorunlu değildir. switch case çok kullanılmasına karşın düşük seviyeli kontrol ifadesidir. Bundan dolayıdır ki yukarıda Java'da yazılan kodu C dilinde yazılamaz. Çünkü C dilinde switch case yapısında char* kabul edilmemektedir. Hatta daha da ilginç **C dilinde switch case yapısı sadece tam sayıları desteklemektedir**. Yukarıdaki

program C dilinde yazılmak isteniyorsa ya if-else yapısı kullanılmalı ya da değerler tam sayıya dönüştürülüp karşılaştırılmalıdır.

Switch case yapısında bir diğer hayati önem taşıyan durum break ifadelerinin mutlaka konulması gerektiridir. Konulmaması durumunda C/C++ ve Java herhangi bir hata vermez fakat break konulmayan case ifadesi çalışması durumunda bir alttaki case ifadesini de kontrol etmeden çalıştıracaktır. Örneğin yukarıdaki kod bloğunda case “Türkiye” kontrol bloğundaki break silinirse Türkiye girildiğinde ekrana yurt içi ve yavru vatan ifadelerinin her ikisi de yazacaktır.

Erişilemeyen Satır Durumu

Java gibi yüksek seviyeli bir dil erişilemeyen satıra izin vermez. Erişilemeyen satır hangi koşulda olursa olsun çalıştıramayacak satırdır. Dolayısıyla yazılmasının bir anlamı yoktur. Örneğin aşağıdaki kodda return altında break kullanılmıştır. Bu satıra hiçbir durumda erişilemez.

```
public class Sayi {
    public static int deger;
    public Sayi(int dgr){
        deger=dgr;
    }
    public int DegerAta(Double yeniDeger){
        String dgr = Double.toString(yeniDeger);
        String ondalik = dgr.substring(dgr.indexOf('.')+1,dgr.length());
        int ondalikKismi = Integer.parseInt(ondalik);
        switch(ondalikKismi)
        {
            case 0: // ondalık kısmı yoktur
                deger = yeniDeger.intValue();
                return deger;
                break; // Erişilemeyen satır
        }
        return 0;
    }
}
```

Fakat aynı durumda C dili hata vermez. Örneğin aşağıdaki kod derlenip çalışacaktır. Fakat bu şekilde bir kod yazımı anlamsız olacağı için kullanılmamalıdır.

```
#include "stdio.h"

int main(){
    int turId;
    printf("Tur girin:");
    scanf("%d",&turId);
    switch (turId) {
        case 1:
            printf("Yonetici");
            return 1;
            break;
        case 2:
            printf("Akademisyen");
            return 2;
```

```

        break;
    case 3:
        printf("Ogrenci");
        return 3;
        break;
    }
    return 0;
}

```

Kontrol bloklarında && || ve ! ifadeleri kullanılabilir. Bunların anlamı && ifadesi ve, || ifadesi veya, ! ifadesi ise değil gösterir. && ifadesinde if bloğunun çalışması için kontrollerden her ikisinin de doğru olması gerekir.

```

#include "stdio.h"

int main(){
    // Girilen sayının pozitif çift sayı olduğunun kontrolü
    int sayi;
    printf("Bir sayi Girin:");
    scanf("%d",&sayi);
    if(sayi % 2 == 0 && sayi >= 0) printf("Girilen sayi pozitif cift sayidir.");
    else printf("Girilen sayi pozitif cift sayi degildir.");
    return 0;
}

```

|| ifadesinde kontrollerden birinin doğru olması if bloğunun çalışmasını sağlar.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("x:");
    int x = in.nextInt();
    System.out.print("y:");
    int y = in.nextInt();
    if(x % 2 == 0 || y % 2 == 0) System.out.println("x * y çifttir");
    else System.out.println("x * y tektir");
}

```

Kontrol ifadeleri boolean türden kontrol yapar ve sonuç true ise çalışır false ise çalışmaz. C dilinde 0 değeri false diğer bütün değerler true olarak kabul edilir. Örneğin aşağıdaki C programı ekrana Merhaba yazacaktır.

```

int main(){
    if(1){
        if(135) printf("Merhaba!");
    }
}

```

Fakat aynı şekilde kullanım Java'da derlenme anında hata verecektir. C boolean türü olmadığı için ekrana yazdırılmazken Java'da aşağıdaki gibi bir kullanımda ifadenin doğru olup olmamasına bağlı olarak ekrana true ya da false yazacaktır.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("x:");
    int x = in.nextInt();
}

```

```

System.out.print("y:");
int y = in.nextInt();
    System.out.print( x > y );    // Ekrana true ya da false yazar.
}

```

Aşağıdaki iki kod farklı yazılmalarına rağmen aynı kontrolleri yapıp aynı çıktıları üretirler. C dilinde kullanıcıdan alınacak sayı double ise "%lf" şeklinde alınmalıdır.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Notunuz:");
    double not = in.nextDouble();
    String harf="";
    if(not >= 90) harf="AA";
    else if(not >= 80) harf = "BA";
    else if(not >= 75) harf = "BB";
    else if(not >= 65) harf = "CB";
    else if(not >= 55) harf = "CC";
    else if(not >= 45) harf = "DC";
    else if(not >= 40) harf = "DD";
    else harf="FF";
    System.out.println("Harf: "+harf);
}

```

```

int main(){
    double notu;
    char* harf;
    printf("Notunuz (0-100):");
    scanf("%lf",&notu);
    if(notu < 40) harf="FF";
    else if(notu < 45) harf="DD";
    else if(notu < 55) harf="DC";
    else if(notu < 65) harf="CC";
    else if(notu < 75) harf="CB";
    else if(notu < 80) harf="BB";
    else if(notu < 90) harf="BA";
    else harf="AA";
    printf("Harf: %s\n",harf);
    return 0;
}

```

Dizideki eleman ve indeks değerleri verilerek ilgili indekste eleman var mı yok mu kontrolü Java ve C dilinde aşağıdaki gibi yapılmaktadır.

| | |
|---|---|
| <p>C Dili</p> <pre> #include "stdio.h" #include "stdlib.h" int main(){ // 10 uzunluğunda int dizisi int *dizi = malloc(10*sizeof(int)); dizi[0]=25; dizi[1]=32; dizi[2]=40; dizi[3]=3; dizi[4]=11; dizi[5]=7; dizi[6]=65; dizi[7]=54; dizi[8]=47; dizi[9]=70; int sayi,indeks; </pre> | <p>Java Dili</p> <pre> public static void main(String[] args) { // 10 uzunluğunda int dizisi Scanner girdi = new Scanner(System.in); int []dizi = new int[10]; dizi[0]=25; dizi[1]=32; dizi[2]=40; dizi[3]=3; dizi[4]=11; dizi[5]=7; dizi[6]=65; dizi[7]=54; dizi[8]=47; dizi[9]=70; int sayi,indeks; System.out.print("Hangi sayiyi arıyorsunuz:"); </pre> |
|---|---|

| | |
|--|---|
| <pre> printf("Hangi sayiyi ariyorsunuz:"); scanf("%d",&sayi); printf("Sayiyi hangi indekste ariyorsunuz:"); scanf("%d",&indeks); if(dizi[indeks] == sayi) printf("Sayi var"); else printf("Sayi yok."); free(dizi); return 0; } </pre> | <pre> sayi = girdi.nextInt(); System.out.print("Sayiyi hangi indekste ariyorsunuz:"); indeks = girdi.nextInt(); if(dizi[indeks] == sayi) System.out.println("Sayi var"); else System.out.println("Sayi yok."); } </pre> |
|--|---|

Üçlü Operatör (? :)

Üçlü operatör tek satırda kontrol ve sonucu yazabilmemizi sağlar.

Kontrol Durumu ? Doğru ise çalışır : Doğru değil ise çalışır;

Örneğin aşağıda üçlü operatör ve if kontrollü iki örnek verilmiştir. Fakat yaptıkları iş aynıdır. Java ve C dillerinde kullanım aynıdır.

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Sayı:");
    int sayi = in.nextInt();
    String sonuc = (sayi % 2 == 0 ? "Sayı çifttir." : "Sayı tektir.");
    System.out.println(sonuc);
}

```

```

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Sayı:");
    int sayi = in.nextInt();
    if(!(sayi % 2 == 0)) System.out.println("Sayı tektir.");
    else System.out.println("Sayı çifttir.");
}

```

C dilindeki Karşılığı

```

int main(){
    printf("Sayı:");
    int sayi;
    scanf("%d",&sayi);
    char* sonuc;
    sonuc = (sayi % 2 == 0 ? "Sayi Cifttir" : "Sayi Tektir");
    printf("%s",sonuc);
    return 0;
}

```

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

Programlama Dillerinin Prensipleri

Lab Notları – 5

1. Döngüler

Bir program yazıldığı vakit bazı durumlarda bir satırın birden çok kez çalıştırılması düşünülebilir. Örneğin ekrana 1’den 100’e kadar sayılar yazılmak isteniyor. Bu durumda hepsini printf kullanarak yazmaya kalkışmak 100 satırı sadece bu işlem için doldurmak anlamına gelir. İşte bu durumlarda döngüler kullanılmalıdır. Farklı yapılarda birçok döngü çeşidi bulunmaktadır. Java ve C dili hemen hemen aynı döngü yapılarını kullanır. Arada ufak farklılıklar bulunmaktadır. Döngüler kontrol bakımından iki türdür. Önce test (**pre-test**) ve sonra test (**post-test**) döngüleri, for ve while döngüleri önce test döngüleridir. Do-while ise sonra test döngüsüne girer.

for Döngüsü

```
int main(){ // Kullanışsız ve anlamsız
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    ...
    ...
    return 0;
}

int main(){ // Kullanışlı ve doğru olanı
    for(int i=1;i<=100;i++) printf("%d\n",i);
    return 0;
}
```

Döngünün kaç kez döneceği belli ise bu durumlarda for döngüsü kullanmak daha uygundur. for döngüsü 3 bölümden oluşur ve 3 bölümünde girilmesi zorunlu değildir. Örneğin aşağıdaki Java kodunda sadece ilk bölüm girilmiştir.

```
public static void main(String[] args) {
    for(int i=0 ; ; ){
        if(i++ == 100) break;
        if(i % 10 == 0) System.out.println(i);
    }
}
```

for döngüsünün içerdiği 3 bölümün görevi aşağıdaki gibi özetlenebilir.

for (**ilklenme yeri bir kez çalışır** ; **Kontrol yeri her döngüde bakılır** ; **Güncelleme yeri her döngüde**)
Bazı programlama dillerinde for döngüsünün özel bir hali olan foreach döngüsü kullanılır (C#). Bu döngü aralık tabanlı bir döngüdür. Ve bir serideki elemanları sıra sıra dolaşmayı sağlar. C++’ın bazı versiyonlarında foreach olarak yazılmasa da aralık tabanlı döngü yapısı oluşturulabilmektedir. Java’da da aynı şekilde foreach kelimesi desteklenmez fakat aralık tabanlı döngü desteklenir. Örneğin aşağıdaki tamsayılar dizisinde kullanıcının girdiği sayı aranmaktadır.

```

public static void main(String[] args) {
    int []SayiDizisi = {15, 22, 41, 65, 35, 54, 100 };
    System.out.print("Aradığınız Sayı:");
    Scanner girdi = new Scanner(System.in);
    int sayi = girdi.nextInt();
    for(int i : SayiDizisi){
        if(i == sayi){
            System.out.println("Sayı Var.");
            break;
        }
    }
}

```

Aynı yapı C dilinde normal for döngüsü kullanılarak aranmış olsaydı aşağıdaki gibi yazılmış olacaktı.

```

int main(){
    int SayiDizisi[] = {15, 22, 41, 65, 35, 54, 100 };
    int sayi;
    printf("Aranan Sayı:");
    scanf("%d",&sayi);
    for(int index=0;index<7;index++){
        if(SayiDizisi[index] == sayi){
            printf("Sayı Var.");
            break;
        }
    }
    return 0;
}

```

while Döngüsü

Bu döngü for döngüsüne benzer şekilde kontrol işlemini başta yapar. Dolayısıyla kontrol yanlış dönerse döngü çalışmaz. C dilinde ve Java'da yapısı aynıdır. Örneğin aşağıdaki program kodunda girilen sayıya kadar tam sayıların toplamı ekrana yazdırılıyor.

```

#include "stdio.h"

int main(){
    int sayi,toplam=0;
    printf("Sayı:");
    scanf("%d",&sayi);
    while(sayi != 0) toplam+=sayi--;
    printf("Toplam:%d\n",toplam);
    return 0;
}

```

do-while Döngüsü

Bu döngüyü diğer döngülerden ayıran özellik, koşul ne olursa olsun mutlaka bir kez çalışacaktır. Bunun nedeni kontrol kısmının döngünün sonunda olmasıdır. Aşağıda asal olmayan bir sayı girilene kadar yapılan kontrolde do-while döngüsü kullanılmıştır.

```

public static void main(String[] args) {
    int sayi;
    Scanner girdi = new Scanner(System.in);
    do{
        System.out.print("Sayı:");
        sayi = girdi.nextInt();
    }while(!new String(new char[sayi]).matches(".*?(.+?)\\1+"));
    System.out.println("Girilen sayı asal değildir.");
}

```

Yukarıdaki kod bloğunda while içindeki kontrol başta biraz karmaşık gelebilir. Bir sayının asal olup olmadığının kontrolü çok farklı şekillerde yapılabilir. Burada regex kullanılarak yapılmıştır. Girilen sayı uzunluğunda boş karakterler dizisi oluşturulup bir String içerisine atılıyor. Daha sonra bu String içerisinde regex kullanılarak bir karşılaştırma yapılıyor. İlk soru işareti sıfır sayısının girilmiş mi kontrol eder. Burada nokta herhangi bir karakter ile eşleşme demektir. + işareti ise bir önceki ifadenin 1 veya daha fazla tekrarlanıp tekrarlanmadığını kontrol eder. 1+ ifadesi ise kendinden önce gelen parantezdeki kısmın 2 veya daha fazla tekrarlanıp tekrarlanmadığına bakar.

(2 veya daha fazla tekrarlanma) x (2 veya daha fazla tekrarlanma) = Asal Olmaz

break ve continue İfadeleri

Döngülerde genel kontrolün dışında bazı durumlar oluşması halinde de döngüden tamamen çıkılmak ya da bir turu es geçmek düşünülebilir. break ifadesi döngüyü koşulsuz bir şekilde sonlandırmayı sağlar. Örneğin aşağıdaki Java kodunda girilen ağırlıklar toplanıyor fakat olurda negatif bir ağırlık girerse döngü sonlandırılıyor.

```

public static void main(String[] args) {
    double ToplamAgirlik=0, agirlik;
    Scanner girdi = new Scanner(System.in);
    do{
        System.out.print("Agirlik:");
        agirlik = girdi.nextDouble();
        if(agirlik < 0) break;
        ToplamAgirlik += agirlik;
    }while(ToplamAgirlik <= 100);
    System.out.println("Girilen Toplam Ağırlık: "+ToplamAgirlik);
}

```

continue ifadesi ise o anki turu es geçmeyi sağlar. Örneğin 3'e bölünenlerin ekrana yazdırıldığı bir programda continue aşağıdaki gibi kullanılabilir. Burada 3'e tam bölünemeyenler es geçilmiştir.

```

int main(){
    for(int i=1;i<=100;i++){
        if(i%3 != 0) continue;
        printf("%d ",i);
    }
    return 0;
}

```

Örneğin bileşik faiz probleminde günlük faiz oranı %0.02 olan bir kredide 20000 TL çekilmek isteniyor fakat 25000 TL'den fazla geri toplam ödeme olması istenmiyor ise kredi kaç günde geri ödenmesi gerekir. Bu problemi döngü kullanarak çözebiliriz.

Bileşik faiz olduğu için her gün faiz hesabına giren miktar değişecektir. Basit formülü

$$\text{Faiz} = (A \times n \times t) / 3600$$

```
public static void main(String[] args) {
    double miktar=20000,faiz_Orani=0.02;
    int t;
    for(t=1;miktar<=25000;t++){
        double faiz = (miktar * faiz_Orani * t)/3600;
        miktar += faiz;
    }
    System.out.println("En fazla 25000 geri ödemek için "+t+" günlük alınabilir.");
}
```

Sonsuz Döngüler

Döngülerdeki mantık doğru olduğu sürece sonlanması hiç gerçekleşmeyecek bir kontrolde döngü sonsuz defa dönecektir. for döngüsü düşünüldüğünde 3 bölümden oluştuğu ve bu bölümlerin girilmesi zorunlu olmadığı için boş bırakılırsa sonsuz döngü olur. Aşağıdaki ilk örnek Java'da diğeri C dilinde verilmiştir.

```
while(true){
    System.out.println(":");
}

for(;;){
    printf(": ");
}
```

Sonsuz döngü kurup içinde break ifadesi ile bu döngüden çıkılabilir. Bunun sıklıkla kullanım örnekleri vardır.

Önemli: Döngü ifadesinin sonuna ; işareti konulmaz. Konulursa bu bir derlenme hatası değildir. Ama bağlı bulunduğu bloğu çalıştırmaz. Örneğin aşağıdaki for döngüsü ekrana 0'dan 9'a kadar yazması beklenirken ekrana sadece 10 yazacaktır. for döngüsü çalışmış fakat noktalı virgül kullanımı nedeniyle bir alt satır döngüye bağlanmamıştır.

```
int main(){
    int i;
    for(i=0;i<10;i++);
        printf("%d ",i);

    return 0;
}
```

Hangi Döngü Kullanılmalı

```
while(kontrol_ifadesi){  
    // Döngü gövdesi  
}
```



```
for( ; kontrol_ifadesi ; ){  
    // Döngü gövdesi  
}
```

```
for(ilkleme; kontrol_ifadesi; güncelleme){  
    // Döngü gövdesi  
}
```



```
ilkleme  
while( kontrol_ifadesi ){  
    // Döngü gövdesi  
    güncelleme  
}
```

Hangi döngü kullanımında programcıya rahatlık sağlıyorsa o döngü kullanılmalıdır. Kaç kere döneceği bilinen bir döngüde for kullanımı beklenir. Bir değer girildiğinde çıkılacak bir döngüde mesela while veya do-while mantıklıdır.

İç içe Döngüler

İç içe döngüler genelde bir dış döngü ve bir veya birden fazla iç döngü içerirler. Her dış döngü bir iterasyon ilerlediğinde iç döngüler baştan başlayarak tekrarlanırlar. En dış döngüde aynı iterasyon tekrarı olmayacaktır. Bu tarz döngüler birden çok dizi boyutundan sıklıkla rastlanırlar. Aşağıdaki kod bloğunda çarpım tablosu ekrana yazdırılmıştır.

```
public static void main(String[] args) {  
    // TODO code application logic here  
    System.out.println("          Çarpım Tablosu");  
    System.out.println("-----");  
    // sayı başlıklarını yaz  
    System.out.print("# | ");  
    for(int i=1;i<=9;i++){  
        System.out.print("  " + i);  
    }  
    System.out.println("\n-----");  
  
    for(int i=1;i<=9;i++){  
        System.out.print(i + " | ");  
        for(int j=1; j<=9; j++){  
            if(i*j <10) System.out.print("  " + (i*j));  
            else System.out.print("  " + (i*j));  
        }  
        System.out.println();  
    }  
}
```

Durum Etiketleri

Kullanımı zorunlu olmamakla birlikte Java’da etiketler kullanılabilir. Bu etiketler break ve continue ile birlikte kullanılarak istenilen yere sıçrama yapılabilir. Java’da **goto** kullanımı olmadığı için bu tarz bir kullanım verilmiştir. Fakat bu tarz bir kullanım da goto ile aynı kapıya çıktığı için kodu spaghetti koda dönüştürecektir. Bu tarz kullanımlardan kaçınılmalıdır. Durum etiketleri C dilinde ancak **goto** ile kullanılabilir. Java’da etiketler mutlaka çağrıldığı yerden yukarıda tanımlanmalıdırlar.

```
public static void main(String[] args) {
    outer:
    for(int i=1;i<=9;i++){
        inner:
        for(int j=1;j<=9;j++){
            if(i*j >= 10) break outer;
            System.out.print(" " + i*j);
        }
    }
    System.out.println();
}
```

```
public static void main(String[] args) {
    outer:
    for(int i=1;i<=9;i++){
        inner:
        for(int j=1;j<=9;j++){
            if(i*j >= 10) continue outer;
            System.out.print(" " + i*j);
        }
    }
    System.out.println();
}
```

C dili için

```
#include "stdio.h"
int main(){
    for(int i=1;i<=9;i++){
        inner:
        for(int j=1;j<=9;j++){
            if(i*j >= 10) goto outer;
            printf(" %d",i*j);
        }
    }
    outer:
    printf("\n");
    return 0;
}
```

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

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 çağırımı 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 metod 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.

| | |
|--|--|
| <pre> // 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; } </pre> | <pre> // 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; } </pre> |
|--|--|

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 yığığında tutulurlar. Fonksiyon çağrısı bittiğinde bellekten silinirler. Java'da ise sınıf dışında metod 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): Çağırana, çağırılana değeri direk gönderir. Formal parametre, asıl parametre ile ilklendiği olur. Örneğin aşağıda daha önce yazılmış olan Sayı sınıfına DegerAta adında bir metod 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 çağırma şekli C++ dilinde desteklenmektedir. Java referans ile çağırma desteklenmez her türlü çağırma değeri ile çağırılmazdır. 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 çağrı türünde parametre türü bir göstericidir ve çağırırken değeryerine adres gönderilir. 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.

```
#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;
}
```

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.

```
#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; // deęişme 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 çağrılar için yer ayrılacak ve yavaşlamaya sebep olacaktır. Aşağıda daha önce Java'da yazılmış olan Sayı sınıfına Faktoriyel metodu ekleniyor bu metod Sayı nesnesinin içinde taşıdığı değerin faktöriyelini döndüren bir metod fakat bu metod Fakt metodunu içeriden çağırıyor ve Fakt metodu aslında özyineleme kullanarak sonucu bulan bir metod.

```

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

```
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 çalıştırılmaz. 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());  
}
```

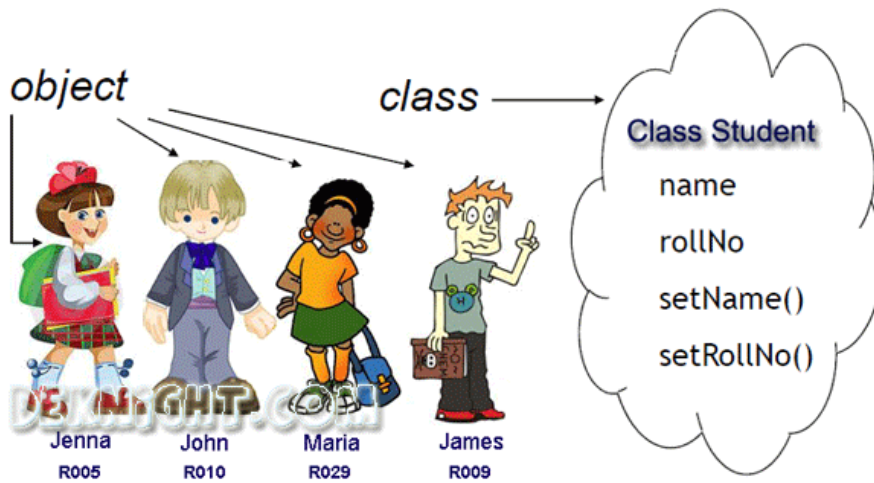
Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

Programlama Dillerinin Prensipleri

Lab Notları – 7

Sınıf Tasarımı

Prosedürel dillerde (Ada, Basic, C) program yazımında veri yapıları ve algoritmaların tasarlanmasını içerir. Fakat C++ ve Java gibi Nesne yönelimli dillerde sınıf ve sınıftan türetilen nesnelerin vermiş olduğu güçle daha kullanılabilir, daha modüler ve gerçeğe kolayca uyarlanabilen programlar yazmak kolaydır. Java ve C++'ın sınıf tasarımına bakıldığında nitelik ve davranış terimlerinin karşılıkları C++'ta niteliklere veri üyeleri (data members), davranışlara ise üyelik fonksiyonları (member functions) karşılık gelir. Java'da ise durum, nitelikler değişkenler (instance, class), davranışlar ise metotlardır. Nesne ile sınıf arasındaki fark nesneler, sınıftan türetilen elemanlardır.



Java'da sınıf tanımı aşağıdaki gibi yapılmaktadır.

```
class sınıf_adi{  
    niteleyici(public, private vb.) tür(int, double vb.) degisken_adi;  
  
    niteleyici(public, private vb.) dönüş_türü(int, double vb.) metot_adi(parametreler){  
        ...  
    }  
}
```

Önemli: Java'da sınıf adı ile sınıfın bulunduğu dosyanın adı aynı olmalıdır. Java'da bir dosyada birden fazla sınıf tanımlanamaz. Ancak içi içe sınıf olabilir.

Aşağıda örnek bir Java sınıf tasarımı görünmektedir.

```
public class Kisi {  
    private String isim;  
    private int yas; // Yıl olarak  
    private float boy; // cm
```

```

private float kilo; // kg
public Kisi(String ad,float by,float kl){
    isim=ad;
    yas=0;
    boy=by;
    kilo=kl;
}
public void Yasllerle(int yil){
    yas+=yil;
    if(yas<18)boy+=1;
}
public void YemekYe(float kalori){
    kilo+= (kalori/1000);
}
}

```

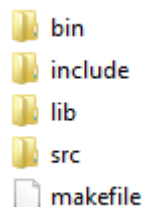
Yapıcı metot sınıf ile aynı adı taşır ve dönüş değeri tanımlanmaz. Bir sınıftan nesne birçok yolla oluşturulabilir. Bunun anlamı birden fazla yapıcı metot olabilir.

```

class Kisi{
...
public Kisi(String ad){
    isim=ad;
    yas=0;
    boy=20;
    kilo=4;
}
...
}

```

C dilinde ise sınıf yapısı desteklenmez. Bu ders kapsamında Java dilinde sınıf tasarımı anlatılırken C dili bu yapıya benzetilmeye çalışılacaktır. Dolayısıyla yukarıda Java’da tasarlanmış olan sınıf, C dilinde benzetilmeye çalışıldığında başlık ve kaynak dosyası iki farklı dosya halinde yazılacak böylelikle başka biri bu yapıyı kullanmak istendiğinde kod gizliliği sağlanmış olacaktır. Bunun için aşağıdaki şekilde verilmiş olan klasör hiyerarşisi kullanılacaktır. Bin klasörü içerisinde çalıştırılabilir program, include klasörü içerisinde başlık dosyalarımız, src klasörü içerisinde başlık dosyalarına ait kaynak dosyaları ve lib klasöründe derlenme sonucu oluşan .o uzantılı output dosyaları olacaktır.



İlk önce Kisi.h isminde bir başlık dosyası aşağıdaki gibi tanımlanır.

```

#ifndef KISI_H
#define KISI_H
#include "stdio.h"

```



```
#include "stdlib.h"

struct KISI{
    char *isim;
    float boy;
    float kilo;
    int yas;
};
typedef struct KISI* Kisi;

Kisi KisiOlustur(char isim[],float,float);
void Yasllerle(const Kisi,int);
void YemekYe(const Kisi,float);
void KisiYazdir(const Kisi);
void KisiYoket(Kisi);

#endif
```

Burada dikkat edilmesi gereken Kisi ifadesi bir KISI göstericisini temsil etmektedir. Yani bir göstericidir. Buradaki metotların normalde KISI yapısı ile bir ilgisi yoktur bundan dolayı yapının elemanlarını değiştirebilmek için Kisi yapısını parametre olarak almaktadır. Bu metotların gövdeleri Kisi.c isimli kaynak dosyasında verilmiştir.

```
#include "Kisi.h"

Kisi KisiOlustur(char isim[],float by,float kl){
    Kisi this;
    this = (Kisi)malloc(sizeof(struct KISI));
    this->isim = isim;
    this->yas=0;
    this->boy = by;
    this->kilo = kl;
    return this;
}
void Yasllerle(const Kisi k,int yil){
    k->yas += yil;
    if(k->yas < 18)k->boy+=1;
}
void YemekYe(const Kisi k,float kalori){
    k->kilo += (kalori/1000);
}
void KisiYazdir(const Kisi k){
    printf("isim:%s\n",k->isim);
    printf("Yas:%d\n",k->yas);
    printf("Boy:%.2f\n",k->boy);
    printf("Kilo:%.2f\n",k->kilo);
}
void KisiYoket(Kisi k){
    if(k == NULL) return;
    free(k);
    k=NULL;
}
}
```

Parametrede Kisi const olarak alınmıştır. Bunun anlamı Kisi yapısı sabittir fakat içerdiği elemanlar değiştirilebilir ama gösterici olarak kendisi bir başka adrese atanamaz. Nesne tasarımına benzetilmesi için bir yapıcı ve yıkıcı metot görevi görecektir KisiOlustur ve KisiYoket metotları tanımlanmıştır. Bu yapıyı test eden kod parçası aşağıda verilmiştir.

```
#include "Kisi.h"

int main(){
    Kisi k = KisiOlustur("Ahmet",25,5);
    Yaslarla(k,3);
    YemekYe(k,500);
    KisiYazdir(k);
    KisiYoket(k);

    return 0;
}
```

Derlemek için gerekli make dosyası aşağıdadır.

```
hepsi: derle calistir

derle:
    gcc -I ./include/ -o ./lib/Kisi.o -c ./src/Kisi.c
    gcc -I ./include/ -o ./bin/Test ./lib/Kisi.o ./src/Test.c

calistir:
    ./bin/Test
```

this Terimi

this terimi Java’da o anda oluşturulan nesneyi ifade etmek için kullanılır. Mesela aşağıdaki örneğe bakıldığında yapıcı metodun parametresi ile kişi sınıfının alt alanı aynı adı taşımakta. Bu derleyici için bir karmaşıklığa sebep olmaktadır. isim=isim; ifadesinde yapılan şey parametre olan isim’in **kendi üzerine atanmasıdır**. Dolayısıyla Kisi sınıfından bir nesne türetip ismini yazmaya kalktığımızda String olduğu için ve değeri verilmemiş olduğu için ekrana **null** yazacaktır. Bunu düzeltmek için this terimi kullanılmalıdır.

```
class Kisi{
public String isim;
...

public Kisi(String isim){
    isim=isim;
    yas=0;
    boy=20;
    kilo=4;
}
...
}

// Doğrusu
public Kisi(String isim){
```

```

    this.isim=isim;
    yas=0;
    boy=20;
    kilo=4;
}

```

C++ ve Java'da C#'ta var olan property tanımlaması yoktur. Property sınıfın sahip olduğu alt alanlara erişmeye yarar. Bu erişme sadece değerini görme olabileceği gibi (get), değerini değiştirme de olabilir (set). Fakat bu görevi Java'da metotlar yazarak yerine getirebilirsiniz. Örneğin yukarıda tanımlanmış olan sınıflardan örnek vermeye devam edersek...

```

class Kisi{
...
    public int Yas(){
        return yas;
    }
    public String Boy(){
        return String.format("%.1f", boy);
    }
    public String Kilo(){
        return String.format("%.1f", kilo);
    }
....
}

```

```

...
void KisiYazdir(const Kisi k){
    printf("isim:%s\n",k->isim);
    printf("Yas:%d\n",Yas(k));
    printf("Boy:%.2f\n",Boy(k));
    printf("Kilo:%.2f\n",Kilo(k));
}
int Yas(const Kisi k){
    return k->yas;
}
float Boy(const Kisi k){
    return k->boy;
}
float Kilo(const Kisi k){
    return k->kilo;
}
...

```

Java'da sınıflardan türetilen nesneler sadece heap bellek bölgesinde bulunabilirler.

Java

```

public static void main(String[] args) {
    Kisi k = new Kisi("Ahmet");
    k.YemekYe(586);
    System.out.println(k.Kilo());
}

```

İç içe Sınıf Tanımı

Java'da içi içe sınıf yazılabilir. Aşağıda bir örnek verilmiştir. Fakat taşınabilirlik açısından Canta sınıfının farklı bir dosyada tek bir sınıf şeklinde yazılması daha doğrudur.

```
public class Kisi {
    private Canta tasidigiCanta;
    ....
    private class Canta{
        private float hacim;
        public Canta(float hacim){
            this.hacim = hacim;
        }
    }
    public void CantaAl(float hacim){
        tasidigiCanta = new Canta(hacim);
    }
    ....
}
```

```
Kisi k = new Kisi("Ali",100,60);
k.CantaAl(25);
```

Yıkıcı Metotlar

Java çöp toplayıcıya sahip bir dil olduğu için ve sınıflardan türetilmiş bütün nesneler heap bellek bölgesinde olduğu için bir nesneye bellekten geri dön komutu gönderilemez. Fakat bir nesne yıkıldığı an bazı işlemler yapılmak isteniyorsa bu durumda finalize metodu kullanılabilir. Bu metot nesne çöp toplayıcı tarafından yıkıldığı zaman çağrılır. Fakat programcının dışarıdan bunu çağırması nesnenin yıkıldığı anlamına gelmez. Aşağıdaki örneği inceleyelim.

```
public class Kisi {
    ....
    protected void finalize() throws Throwable {
        try {
            // açık dosya varsa kapat
            System.out.println("Çağrıldı");
        }
        finally {
            super.finalize();
        }
    }
}
```

```
public static void main(String[] args) {
    Kisi k = new Kisi("Ali",100,60);
    try{
        k.finalize();
    }
    catch(Throwable t){

    }
    k.Yasllerle(15);
}
```

Yukarıda görüldüğü gibi Kişi sınıfından türetilen k nesnesi yıkılması için finalize metodu çağrılmıştır. Ama daha sonra Yaslarla metodu çağırılmış ve k nesnesi hayatını sürdürmeye devam etmiştir.

Fakat C dilinde durum tamamen farklıdır. C programlama dilinde heap bellek bölgesinin kontrolü programcının elinde olduğu için o bölgede işi bittiğinde geri döndürmelidir. C dilinde sınıf desteği olmadığı için yıkıcı metod bulunmamaktadır fakat Heap bellek bölgesinde açılan bir alan free komutu ile belleğe geri iade edilmelidir.

Erişim Niteleyicileri

Java için public, protected ve private niteleyicileri bulunmaktadır. Bu niteleyiciler sınıfın elemanlarının görünürlüğünü ayarlamaktadır. Java için bakıldığı zaman aşağıdaki tablo konunun anlaşılması için yardımcı olacaktır.

| Niteleyici | Aynı Sınıftan erişim | Aynı Paketten erişim | Alt Sınıftan Erişim (Kalıtım) | Farklı Paketten Erişim |
|-------------------|----------------------|----------------------|-------------------------------|------------------------|
| public | VAR | VAR | VAR | VAR |
| protected | VAR | VAR | VAR | YOK |
| varsayılan | VAR | VAR | YOK | YOK |
| private | VAR | YOK | YOK | YOK |

Önemli: Şu ana kadar bahsedilen niteleyicilere sınıf dışından erişilebiliyorsa ve programcı bunlara erişmek istiyorsa mutlaka sınıftan bir nesne türetmelidir. Fakat bazı durumlarda sınıftan nesne türetilmeden kullanılmak istenebilir veya zorunda kalınabilir bu durumda bir başka niteleyici olan static devreye girer. Örneğin sınıfsız program yazılamayan Java'da main programı başlatmak için işletim sistemi tarafından çağrılır ve çağrıldığında nesne türetilmediği için hata vermesi beklenir fakat hata oluşmasını engelleyen başındaki static ifadesidir.

```
public static void main(String[] args) {  
    // TODO code application logic here  
    ....  
}
```

Başlık Dosyaları

C++ derleyicisi kendi başına tanımlamaları arayıp bulma yeteneğinden yoksundur. Dolayısıyla programcının bu tanımlamaları derlenme ve link işlemlerinde derleyiciye göstermesi gerekmektedir. Bir programcının tasarlamış olduğu bir aracı (tool) bir başka programcı da kullanacaktır. Fakat burada tanımlamaları verme zorunluluğu bulunduğu için ama diğer tarafta da kod gizliliği olduğu için bunu ancak başlık dosyaları ile sağlayabilir. Başlık dosyaları tanımlamaları (imzaları) verirken gerçekleştirim (fonksiyon gövdelerini) vermez. Bu şekilde hem kod gizlenmiş hem de derleyiciye imzalar yardımıyla tanımlamalar verilmiş olur. C dilinde nesne yönelimli tasarım bulunmamakta fakat başlı dosyaları tanımlanabilmektedir. Başlık dosyalarının uzantıları .h şeklindedir.

Arac.h

```
#ifndef ARAC_H  
#define ARAC_H  
  
#include "stdio.h"  
#include "stdlib.h"
```

```

struct ARAC{
    float hiz; //km/saat
    int yil; // Model yılı
};
typedef struct ARAC* Arac;
Arac AracOlustur(int);
void Hizlan(const Arac,float);
void Yavasla(const Arac,float);
float Hiz(const Arac);
int Yil(const Arac);
void AracYoket(Arac);

#endif

```

Arac.c

```

#include "Arac.h"

Arac AracOlustur(int yil){
    Arac this;
    this = (Arac)malloc(sizeof(struct ARAC));
    this->yil = yil;
    this->hiz=0;
    return this;
}
void Hizlan(const Arac a,float hz){
    a->hiz += hz;
}
void Yavasla(const Arac a,float hz){
    a->hiz += hz;
}
float Hiz(const Arac a){
    return a->hiz;
}
int Yil(const Arac a){
    return a->yil;
}
void AracYoket(Arac a){
    if(a == NULL) return;
    free(a);
    a=NULL;
}

```

Test.c (Geliştirilmiş olan aracı kullanan program)

```

#include "Arac.h"

int main(){
    Arac ar = AracOlustur(2017);
    Hizlan(ar,50);
    printf("Suanki Hizi: %.2f\n",Hiz(ar));
    printf("Model Yili: %d",Yil(ar));
    AracYoket(ar);
    return 0;
}

```

Java'da bu anlamdaki kullanım C/C++ dillerinden farklıdır. Java'da başlık dosyaları yoktur hatta ihtiyaçta yoktur. Bir sınıf tasarladığınız zaman bunu derlersiniz ve bu sınıfı dışarıya verebileceğiniz bir .jar dosyası oluşur. Sizin sınıfı kullanacak kişi bu jar dosyasını projesine ekleyip gerekli yerlerde import etmesiyle zaten kullanabilecektir. C++ derlerken oluşan .o dosyaları Java'da yerini .class dosyalarına bırakır. C/C++'ta başlık dosyaları include edilirken, Java'da paketler import edilir.

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

Nesne Yönelimli Programlama

Kalıtım: Tasarlanan yeni türlerin tamamen bağımsız yeni türler olduğu düşünülemez. Mutlaka daha önce oluşturulmuş bazı türlerle benzerlikleri bulunur. Bu durumda benzer olan o özellikleri yeniden tanımlamak yerine, var olanları kullanıp kalıttan faydalanarak sadece yeni olan özellikleri yazmak zaman ve maliyetten kazanç sağlayacaktır.

Örnek Java : Kalıtımın desteklendiği Java programlama dilinde bir sıralama yapan sınıf yazıldı. Buna ek olarak sayıların ortalamasının da bulunduğu bir sınıf yazılmak isteniyor. Bu durumda yeni bir sınıf yazarak sıralama yapan sınıftan kalıtım alınırsa sayılar protected olarak tanımlandığı için kalıtlayan sınıflar bu sayılara erişebilir. Dolayısıyla ortalamayı da rahatlıkla bulacaktır. Java’da kalıtım denildiğinde **subclass** ve **superclass** ifadeleri göze çarpar. Aşağıdaki örnekte subclass Mean sınıfı, superclass ise Order sınıfı olmaktadır. Yani kalıtım alınan sınıf superclass kalıtlayan sınıf ise subclass’tır.

```
package SINIFLAR;

/**
 *
 * Order.Java
 */
public class Order {
    protected int[] Sayilar;
    public Order(int []sayilar){
        Sayilar = new int[sayilar.length];
        for(int i=0;i<sayilar.length;i++){
            Sayilar[i] = sayilar[i];
        }
    }
    public Order(int sayi){
        Sayilar = new int[String.valueOf(sayi).length()];
        int indeks=Sayilar.length-1;
        while(sayi > 0) {
            Sayilar[indeks--] = sayi % 10;
            sayi /= 10;
        }
    }
    public int[] Sirala(){
        for (int i = 0; i < Sayilar.length - 1; i++)
        {
            for (int j = 1; j < Sayilar.length - i; j++)
            {
                if (Sayilar[j] < Sayilar[j - 1])
                {
                    int tmp = Sayilar[j - 1];
                    Sayilar[j - 1] = Sayilar[j];
                    Sayilar[j] = tmp;
                }
            }
        }
        int []sirali = new int[Sayilar.length];
        for(int i=0;i<Sayilar.length;i++){
            sirali[i] = Sayilar[i];
        }
        return sirali;
    }
    @Override
    public String toString() {
```



```

        String ekran="";
        Sirala();
        for(int sayi : Sayilar){
            ekran += sayi + " ";
        }
        return ekran;
    }
}

package SINIFLAR;

/**
 *
 * Mean.Java
 */
public class Mean extends Order{

    public Mean(int[] sayilar) {
        super(sayilar);
    }
    public double Ortalama(){
        double toplam=0;
        for(int sayi : Sayilar){
            toplam+=sayi;
        }
        return toplam/Sayilar.length;
    }
    @Override
    public String toString() {
        return super.toString() + "\nOrtalama: "+Ortalama();
    }
}

```

Yukarıda sadece Mean sınıfı kullanılarak hem sıralama hem de ortalama alınabilir. Sıralama metodu public tanımlandığı için Mean sınıfının da bir alt alanı olacaktır. Burada çok dikkat edilmesi gereken yer Mean nesnesi oluşturduğu zaman arka planda Order nesnesi de oluşacağı için onun yapıcı fonksiyonları kontrol edilmelidir. Örneğin Order sınıfının sadece bir yapıcı metodu vardır ve parametre almaktadır. Bu durumda Mean nesnesi oluşunca bu parametre sağlanmalıdır. Bu da super(sayilar) şeklinde Mean sınıfında sağlanmıştır. Eğer parametre sağlama gerekmiyorsa bu durumda super() şeklinde bir ifadeyi yazmaya gerek yoktur. Çünkü üst sınıfın yapıcı metodu her türlü çağrılmaktadır.

Yukarıdaki örnek tekli kalıtıma (single inheritance) örnektir. Çünkü sadece bir sınıftan kalıtlamıştır. Fakat birden çok sınıftan kalıtlamayı destekleyen programlama dilleri de vardır. Örneğin C++ çoklu kalıtımı (multiple inheritance) destekler. Fakat Java çoklu kalıtımı desteklemez. Java'da çoklu kalıtıma uyacak bir yapı tasarlanmak isteniyorsa yapılacak işlem arayüz tasarımı yapıp arayüz üzerinden kalıtımı gerçekleştirmek. Çünkü Java **çoklu ara yüz** kalıtımını desteklemektedir.

Overload ve Override Arasındaki Fark

Overload bir metoda yeni anlamlar kazandırarak çok biçimliliği sağlamaktır. Böylelikle bir metod birden fazla şekilde çağrılabilir.

Override ise metodu yeniden tanımlamaktır. Dolayısıyla eski metodu ezmiş olacaktır. Bu şekilde metod sadece bir yolla çağrılır. Örneğin yukarıdaki kod bloğuna gidersek Order sınıfının iki adet yapıcı metodu vardır. Bu Order nesnesi iki farklı şekilde oluşabileceğini gösterir. Yani Overload'a örnektir.

Diğer taraftan Mean sınıfında toString metodu override edilmiş eğer edilmeseydi ve mean nesnesi ekrana yazdırılmak istenseydi Order sınıfındaki toString metodu çağrılacaktı ama Mean sınıfı içinde tekrar yazıldığı için Order sınıfındaki toString metodunu ezmiştir. C dilinde fonksiyon overload veya override özellikleri desteklenmez.

C Dilinde Kalıtımın Benzetilmesi

Yukarıda Java dilinde yazılan örnek C dili için yazılmak istenirse geçen hafta bahsedilen benzetme yöntemi kullanılacak bu hafta da kalıtımın nasıl benzetilebileceği anlatılacaktır. İlk başta Order.h ve Order.c dosya içerikleri aşağıda verilmiştir.

```
#ifndef ORDER_H
#define ORDER_H
#include "stdio.h"
#include "stdlib.h"

struct ORDER{
    int *Sayilar;
    int uzunluk;
    int* (*Siralama)(struct ORDER*);
    void (*Yaz)(struct ORDER*);
    void (*Yoket)(struct ORDER*);
};
typedef struct ORDER* Order;

Order OrderOlustur(int sayilar[],int);
Order OrderOlustur_I(int,int);
int* _Siralama(const Order);
void EkranaYaz(const Order);
void OrderYoket(Order);

#endif

#include "Order.h"

Order OrderOlustur_I(int sayi,int uzunluk){
    Order this;
    this = (Order)malloc(sizeof(struct ORDER));
    this->Sayilar = malloc(sizeof(int)*uzunluk);
    this->uzunluk = uzunluk;
    int indeks=uzunluk-1;
    while(sayi > 0) {
        this->Sayilar[indeks--] = sayi % 10;
        sayi /= 10;
    }
    this->Siralama=&_Siralama;
    this->Yaz=&EkranaYaz;
    this->Yoket=&OrderYoket;
    return this;
}

Order OrderOlustur(int sayilar[],int uzunluk){
    Order this;
    this = (Order)malloc(sizeof(struct ORDER));
    this->Sayilar = malloc(sizeof(int)*uzunluk);
    this->uzunluk = uzunluk;
    int i;
    for(i=0;i<uzunluk;i++){
        this->Sayilar[i] = sayilar[i];
    }
    this->Siralama=&_Siralama;
    this->Yaz=&EkranaYaz;
    this->Yoket=&OrderYoket;
    return this;
}

int* _Siralama(const Order order){
    int i,j;
```

```

        for (i = 0; i < order->uzunluk - 1; i++){
            for (j = 1; j < order->uzunluk - i; j++){
                if (order->Sayilar[j] < order->Sayilar[j - 1]){
                    int tmp = order->Sayilar[j - 1];
                    order->Sayilar[j - 1] = order->Sayilar[j];
                    order->Sayilar[j] = tmp;
                }
            }
        }
        return order->Sayilar;
    }
}

void EkranaYaz(const Order order){
    order->Siralas(order);
    int i;
    for (i = 0; i < order->uzunluk; i++){
        printf("%d ",order->Sayilar[i]);
    }
    printf("\n");
}

void OrderYoket(Order order){
    if(order == NULL) return;
    free(order->Sayilar); // dizi de Heap'te oluşturulmuştu
    free(order);
    order=NULL;
}

```

Struct içerisinde fonksiyon tanımlanamaz fakat **fonksiyon göstericisi** tanımlanabilir. Burada geçen haftadan farklı olarak yapı içerisinde fonksiyon göstericisi tanımlanmış ve yapının oluşturulduğu fonksiyon içerisinde ilgili fonksiyonlara göstericiler atanmıştır.

C dilinde Kalıtımın benzetilmesi için aşağıdaki gibi Mean yapısı içerisine Order yapısından bir değişken tanımlanmalıdır. İsmi Java diline benzetilmesi için super olarak verilmiştir.

```

#ifndef MEAN_H
#define MEAN_H
#include "Order.h"

struct MEAN{
    Order super;
    double (*Ortalama)(struct MEAN*);
    void (*Yaz)(struct MEAN*);
    void (*Yoket)(struct MEAN*);
};
typedef struct MEAN* Mean;

Mean MeanOlustur(int sayilar[],int);
double _Ortalama(const Mean);
void Yazdir(const Mean);
void MeanYoket(Mean);
#endif

#include "Mean.h"

Mean MeanOlustur(int sayilar[],int uzunluk){
    Mean this;
    this = (Mean)malloc(sizeof(struct MEAN));
    this->super = OrderOlustur(sayilar,uzunluk);
    this->Ortalama = &_Ortalama;
    this->Yaz = &Yazdir;
    this->Yoket = &MeanYoket;
    return this;
}

```

```

double _Ortalama(const Mean m){
    double toplam=0;
    int i;
    for(i=0;i<m->super->uzunluk;i++){
        toplam += m->super->Sayilar[i];
    }
    return toplam/m->super->uzunluk;
}
void Yazdir(const Mean m){
    m->super->Yaz(m->super);
    printf("\n");
    printf("Ortalama:%.2lf\n",m->Ortalama(m));
}
void MeanYoket(Mean m){
    if(m == NULL)return;
    if(m->super != NULL) m->super->Yoket(m->super);
    free(m);
    m=NULL;
}

```

Tekrar aynı fonksiyonlar ve alt elemanlar tanımlanmamış ve kalıtıma benzer olarak super değişkeni yardımıyla Order yapısının elemanlarına erişilmiştir. Bu yapıları test eden kod aşağıda görülebilir.

```

#include "Mean.h"

int main(){
    int dizi[]={16,95,1,18,3,7,10};
    Mean mean = MeanOlustur(dizi,7);
    mean->Yaz(mean);
    mean->Yoket(mean);
    return 0;
}

```

Tabi burada atlanmaması gereken Java'daki kalıtıma benzer olarak C dilindeki test programında mean değişkeninin gösterdiği yapı heap bellek bölgesinde oluşmakta ve içerisindeki super Order göstericisinin de Heap bellek bölgesinde aynı anda oluşmasıdır. Dolayısıyla Mean değişkenini yokeden fonksiyona dikkatli bakıldığında super'ın gösterdiği bellek bölgesinin de iade edildiği görülecektir. Bu yapının derlenmesi için gerekli make dosyası içeriği aşağıda verilmiştir.

hepsi: derle calistir

derle:

```

gcc -I ./include/ -o ./lib/Order.o -c ./src/Order.c
gcc -I ./include/ -o ./lib/Mean.o -c ./src/Mean.c
gcc -I ./include/ -o ./bin/Test ./lib/Mean.o ./lib/Order.o ./src/test.c

```

calistir:

```
./bin/Test
```

Nesne Karşılaştırması

Java'da Sınıflardan üretilen nesneler aslında heap bellek bölgesinde oluşan yerlerini referans olarak gösterirler. Eğer bu nokta gözden kaçırılırsa aşağıdaki gibi bir karşılaştırmada beklenenin tam tersi bir sonuç alınacaktır.

```

public class Sayi {
    private int deger;
    public Sayi(int dgr){
        deger=dgr;
    }

    @Override
    public String toString() {

```

```
        return String.valueOf(deger);
    }
}
```

```
public static void main(String[] args) {
    Sayi s1 = new Sayi(100);
    Sayi s2 = new Sayi(100);
    if(s1 == s2) System.out.println("Sayılar eşittir");
    else System.out.println("Sayılar eşit değil");
}
```

Yukarıdaki kod bloğunda beklenenin dışında bir sonuç alınmasının sebebi nesnelerin içindeki değerlerin değil de referans karşılaştırması yapılmış olmasıdır. Bunun yerine Sayı sınıfını aşağıdaki gibi yazmak istenen sonucu verecektir.

```
public class Sayi {
    private int deger;
    public Sayi(int dgr){
        deger=dgr;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) { // Karşılaştırılan nesne null olmamalı
            return false;
        }
        if (getClass() != obj.getClass()) { // aynı sınıf nesneleri olmalı
            return false;
        }
        final Sayi sy = (Sayi) obj;

        return this.deger == sy.deger;
    }

    @Override
    public int hashCode() {
        return super.hashCode();
    }

    @Override
    public String toString() {
        return String.valueOf(deger);
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    Sayi s1 = new Sayi(100);
    Sayi s2 = new Sayi(100);
    if(s1.equals(s2)) System.out.println("Sayılar eşittir");
    else System.out.println("Sayılar eşit değil");
}
```

Burada açıklanması gereken metotlar equals ve hashCode metotlarıdır. Her ikisi de Object sınıfında bulunduğu için override yapılarak kendi sınıfımıza yeniden tanımlamışızdır. equals metodu iki nesnenin nasıl eşit olabileceğinin cevabıdır. hashCode metodu ise Java'da bazı koleksiyon sınıflarında örneğin Hash Tablosu, nesnenin hangi yere koyulacağı hashCode döndürdüğü değer ile belirlenir. Eğer hashcode iyi yazılmaz ise aynı iki nesnenin hashcode farklı dönebilme ihtimalinden dolayı iki nesne eşit olmadığı kabul edilip hata yapılacaktır.

Önemli: Hashcode'larının aynı olması nesnelerin aynı olduğu anlamına gelmeyeceği gibi farklı nesnelerin farklı hashcode'lara sahip olma gibi bir zorunluluğu da yoktur. **Fakat equals metodunun true döndüğü nesneler mutlaka aynı hashcode'a sahip olmalıdırlar.**

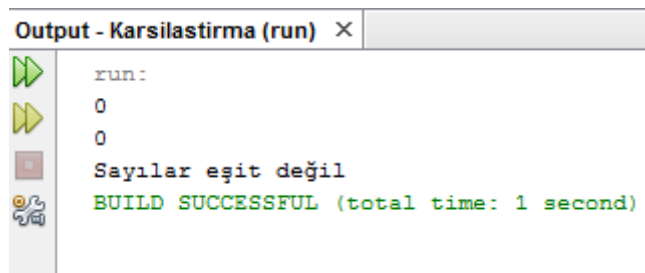
Aşağıdaki hashCode metodu incelendiğinde nesnelerin hash değerleri sayının 101 sayısına bölümünden kalan değer ile belirleniyor. Dolayısıyla 101 ve 0 değerlerine sahip olan iki farklı sayı nesnesi aynı hash değere sahip olacaklardır. Fakat bu onların aynı nesne olduklarını göstermez. Zaten equals metodu ile test yapıldığında farklı sayılar olduğu görülecektir.

```
...
@Override
public int hashCode() {
    return deger % 101;
}
...

public static void main(String[] args) {
    Sayi s1 = new Sayi(0);
    Sayi s2 = new Sayi(101);
    System.out.println(s1.hashCode());
    System.out.println(s2.hashCode());

    if(s1.equals(s2)) System.out.println("Sayılar eşittir");
    else System.out.println("Sayılar eşit değil");
}
```

Ekran Çıktısı



```
Output - Karsilastirma (run) X
run:
0
0
Sayılar eşit değil
BUILD SUCCESSFUL (total time: 1 second)
```

== operatörü yerine equals kullanmamızın nedeni Java'da operator overloading yoktur. Dolayısıyla == operatörü kullanıldığı sürece referans karşılaştırması yapacaktır. Fakat C++'ta durum aynı değildir. C++ operator overloading olduğu için == operatörü yeniden tanımlandığı zaman yapılan karşılaştırma doğru bir karşılaştırma olacaktır. C dilinde de operatör overloading olmadığı için Java tarzı bir yazıma benzetilebilir.

```
#ifndef SAYI_H
#define SAYI_H
#include "stdio.h"
#include "stdlib.h"
typedef enum BOOL { false, true}bool;
struct SAYI{
    int deger;
    bool (*equals)(struct SAYI*,struct SAYI*);
    void (*Yoket)(struct SAYI*);
};
typedef struct SAYI* Sayi;

Sayi SayiOlustur(int);
bool Equals(const Sayi,const Sayi);
void SayiYoket(Sayi);
#endif

#include "Sayi.h"

Sayi SayiOlustur(int dgr){
    Sayi this;
    this = (Sayi)malloc(sizeof(struct SAYI));
    this->deger = dgr;
    this->equals = &Equals;
```

| |
|---|
| <pre> this->Yoket = &SayiYoket; return this; } bool Equals(const Sayi sol,const Sayi sag){ if(sol == NULL sag == NULL) return false; if(sol->deger == sag->deger)return true; else return false; } void SayiYoket(Sayi s){ if(s == NULL) return; free(s); s=NULL; } </pre> |
| <pre> #include "Sayi.h" int main(){ Sayi s1 = SayiOlustur(100); Sayi s2 = SayiOlustur(100); if(s1 == s2) printf("Sayilar esit\n"); //Adres karşılaştırması else printf("Sayilar esit degil\n"); //Değer Karşılaştırması if(s1->equals(s1,s2)) printf("Sayilar esit\n"); else printf("Sayilar esit degil\n"); s1->Yoket(s1); s2->Yoket(s2); return 0; } </pre> |
| <p>hepsi: derle calistir</p> <p>derle:</p> <pre> gcc -I ./include/ -o ./lib/Sayi.o -c ./src/Sayi.c gcc -I ./include/ -o ./bin/Test ./lib/Sayi.o ./src/Test.c </pre> <p>calistir:</p> <pre> ./bin/Test </pre> |

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

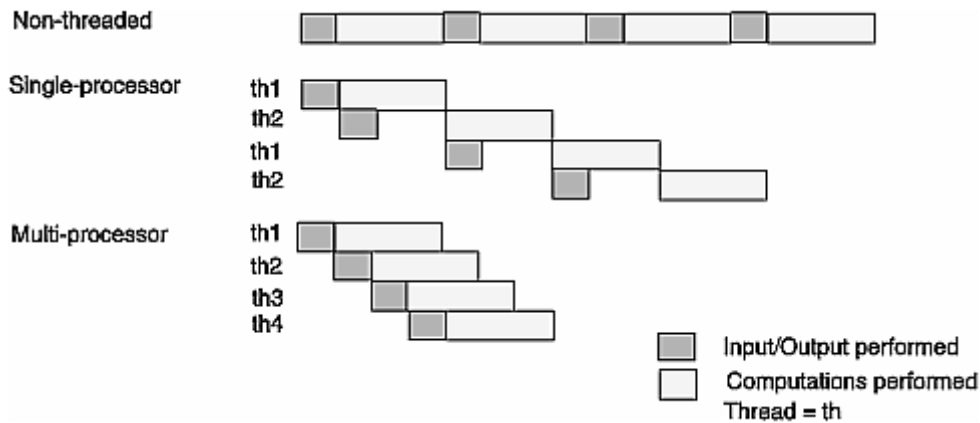
Programlama Dili Prensipleri

Lab Notları – 11

Java'da Thread Mantığı

Bu ders kapsamında thread konusu olarak sadece Java'da thread yapısından bahsedilecektir. C dilinde thread konusu bu ders kapsamında dâhil değildir.

Java'da thread yardımıyla işlemlerin paralel hesaplanması sağlanabilir. Her bir thread bir işlemi yüklenir yürütür ve bitirir. Aşağıdaki resimde eğer thread kullanılmaz ise işlemler arka arkaya dizilir ve işlemci sırayla hepsini işletir. Tek çekirdekli bir işlemcide threadler sahip oldukları işin bir kısmı yapılır daha sonra diğer thread'e geçer. Çok çekirdekli bir işlemcide threadler aynı anda başlayıp aynı zamanda işlem görebilirler.



Java'da işlemler birer nesnedirler. Java'da işlem oluşturmak için öncelikle o işlemin sınıfını yazmak gerekir. Bu sınıf **Runnable** ara yüzünden kalıtım almak zorundadır. Örneğin ekrana karakter yazan bir sınıf tasarımı aşağıda görülmektedir. Runnable ara yüzünden kalıtılayan her sınıf run metodunu override etmek zorundadır. Bu metot sisteme neyin çalıştırılacağını söyler.

```
package arayuz;

/**
 *
 * @author M.Fatih
 */
public class KarakterIslem implements Runnable {
    private char yazilanKarakter;
    private int kacKere;

    public KarakterIslem(char c,int x){
        yazilanKarakter=c;
        kacKere=x;
    }

    @Override
    public void run() {
```



```

        for(int i=0;i<kacKere;i++)
        {
            System.out.print(yazilanKarakter);
        }
        System.out.println();
    }
}

package arayuz;

/**
 *
 * @author M.Fatih
 */
public class RakamIslem implements Runnable{
    private int sonSayi;

    public RakamIslem(int sayi){
        sonSayi=sayi;
    }

    @Override
    public void run() {
        for(int i=1;i<=sonSayi;i++){
            System.out.print(" "+i);
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    // TODO code application logic here

    // İşlemler tanımlanıyor
    Runnable aYaz = new KarakterIslem('a',100);
    Runnable bYaz = new KarakterIslem('b',100);
    Runnable Yaz100 = new RakamIslem(100);

    // Thread oluşturuluyor
    Thread thread1 = new Thread(aYaz);
    Thread thread2 = new Thread(bYaz);
    Thread thread3 = new Thread(Yaz100);

    // Threadler başlatılıyor
    thread1.start();
    thread2.start();
    thread3.start();
}

```

Thread Sınıfı yield Metodu

yield metodu geçici olarak diğer threadlere zaman verir. Örneğin yukarıdaki sayı yazan sınıf yield ile tekrar düzenlenirse her sayı yazdıktan sonra birkaç karakter yazacaktır.

```
@Override
public void run() {
    for(int i=1;i<=sonSayi;i++){
        System.out.print(" "+i);
        Thread.yield();
    }
    System.out.println();
}
```

Thread Sınıfı sleep Metodu:

Belirtilen süre zarfında thread'i uykuya alır. Böylelikle diğer threadler onun yerine çalışmaya başlar. Örneğin yine sayı sınıfı aşağıdaki gibi düzenlenirse

```
@Override
public void run() {
    try{

        for(int i=1;i<=sonSayi;i++){
            System.out.print(" "+i);
            if(i >= 50)Thread.sleep(1); //1 milisaniye uyu
        }
        System.out.println();
    }
    catch(InterruptedException ex){

    }
}
```

Thread Havuzu

Bir önceki başlıkta her bir işlem için bir thread oluşturmuştuk fakat devasa bir programda çok fazla sayıda işlemler olacaktır. Her bir işlem için bir thread tanımlamak verimi oldukça düşürecektir. Bunun yerine sayısını bizim belirtebileceğimiz bir thread havuzu oluşturmak daha mantıklıdır.

```
public static void main(String[] args) {

    // Havuzda 3 adet thread oluşturuluyor
    ExecutorService havuz = Executors.newFixedThreadPool(3);

    // işlemler havuza gönderiliyor
    havuz.execute(new KarakterIslem('a',100));
    havuz.execute(new KarakterIslem('b',100));
    havuz.execute(new RakamIslem(100));

    // Havuzu kapat
    havuz.shutdown();
}
```

Yukarıdaki kod bloğunda havuzdaki thread sayısını 1 yapsaydık o zaman sıralı bir şekilde işlemleri çalıştırmış olacaktık.

Thread Senkronizasyonu

Paylaşılan bir kaynağa farklı thread'ler tarafından aynı anda erişim olursa sistem çökmeleri yaşanabilir. Senkronizasyon içermeyen aşağıdaki örneği inceleyelim.

```
public class Hesap {
    private int toplamPara = 0;
    public int ToplamPara(){
        return toplamPara;
    }
    public void Arttir(int miktar){
        int yeniToplam = toplamPara + miktar;

        try{
            // Veri çöküşünü daha iyi görebilmek için bekleme verildi.
            Thread.sleep(1);
        }
        catch(InterruptedException ex){
        }
        toplamPara = yeniToplam;
    }
}
```

```
public class ParaEkle implements Runnable{
    private Hesap hesap;
    public ParaEkle(Hesap hesap){
        this.hesap = hesap;
    }
    @Override
    public void run() {
        hesap.Arttir(1);
    }
}
```

```
import java.util.concurrent.*;
```

```
public class Deneme {
    public static void main(String[] args) {
        Hesap hesap = new Hesap();
        ExecutorService havuz = Executors.newFixedThreadPool(3);

        // işlemleri oluştur
        for(int i=0;i<100;i++){
            havuz.execute(new ParaEkle(hesap));
        }
        havuz.shutdown();
        // Bütün işlemler bitene kadar bekle
        while(!havuz.isTerminated()){
```

```

    }
    System.out.println("Toplam Miktar:"+hesap.ToplamPara());
}
}
}

```

Yukarıdaki programda senkronizasyon ayarlanmadığı için beklenen çıktı elde edilemeyecektir. Program çalıştığında 3 thread oluşturulan 100 işlemi yapmaya çalışıyor ve hepsi aynı işi yani parayı 1 arttırma işini yapıyorlar ve beklenen değer 100 iken ekran çıktısına bakıldığında 100'den farklı ve tahmin edilemeyen değerler üretecektir.

Bu problemi çözmek için önemli olan bölgeyi kritik bölge olarak belirleyip bu bölgeye aynı anda sadece 1 thread'in girmesine izin vermek. Bunun için yukarıdaki kodu aşağıdaki şekle sokuyoruz.

```

public class Hesap {
    private int toplamPara = 0;
    private final Lock bolge = new ReentrantLock();
    public int ToplamPara(){
        return toplamPara;
    }
    public void Arttir(int miktar){
        bolge.lock(); //Kritik bölge başlangıç
        int yeniToplam = toplamPara + miktar;

        try{
            // Veri çöküşünü daha iyi görebilmek için bekleme verildi.
            Thread.sleep(1);
        }
        catch(InterruptedException ex){
        }
        toplamPara = yeniToplam;
        bolge.unlock(); //Kritik bölgeyi kapat.
    }
}

```

Bu düzenleme yapıldıktan sonra çalıştırıldığında her thread o bölgeye tek başına girecek ve değeri bir arttıracak ve sonuçta ekrana yazılan değer 100 olacaktır.

Basit Matris Çarpımının Thread ile Gerçekleştirimi

Aşağıdaki program değerleri rastgele atanmış iki matrisi çarpıp sonucu ekrana basmaktadır. Burada boyut küçük bir değer girildiğinde thread sayısı arttıkça hesaplama süresi artacaktır. Fakat çok büyük boyutlu 1000x1000 gibi matrislerin çarpımlarında belli sayıya kadar threadi arttırmak hesaplama süresini azaltır.

```

public class Matris {
    public int dizi[][];

    public Matris(Random rnd,int boyut){

```

| |
|---|
| <pre> dizi = new int[boyut][boyut]; for(int i=0;i<boyut;i++){ for(int j=0;j<boyut;j++){ dizi[i][j] = rnd.nextInt(10); } } } public Matris(int boyut){ dizi = new int[boyut][boyut]; } @Override public String toString() { String ekran=""; for(int satir = 0 ; satir < dizi.length; satir++) { for (int sutun = 0 ; sutun < dizi.length; sutun++) { ekran += "\t" + dizi[satir][sutun]; } ekran += "\n"; } return ekran; } } </pre> |
| <pre> public class Carpma implements Runnable { private int satir; private int sutun; private Matris A; private Matris B; private Matris Sonuc; public Carpma(int satir, int sutun, Matris A, Matris B, Matris sonuc) { this.satir = satir; this.sutun = sutun; this.A = A; this.B = B; this.Sonuc = sonuc; } @Override public void run() { for(int i = 0; i < B.dizi.length; i++) { Sonuc.dizi[satir][sutun] += A.dizi[satir][i] * B.dizi[i][sutun]; } } } </pre> |
| <pre> public class JavaApplication17 { </pre> |

```

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    int boyut=3;
    Random rand = new Random();

    Matris sonuc = new Matris(boyut);

    Matris A = new Matris(rand, boyut);
    Matris B = new Matris(rand, boyut);

    ExecutorService havuz = Executors.newFixedThreadPool(8);
    long baslangic = System.nanoTime(); //hesaplama başlıyor

    for(int satir = 0 ; satir < boyut; satir++)
    {
        for (int sutun = 0 ; sutun < boyut; sutun++ )      {
            havuz.execute(new Carpma(satir, sutun, A, B, sonuc));
        }
    }
    havuz.shutdown();

    while(!havuz.isTerminated()){ }

    long bitis = System.nanoTime(); //hesaplama bitiyor
    double sure = (bitis-baslangic)/1000000.0;

    // A Matrisi yazdırılıyor
    System.out.println(" A Matrix : ");
    System.out.println(A);
    // B Matrisi yazdırılıyor
    System.out.println(" B Matrix : ");
    System.out.println(B);
    // Sonuç Matrisi yazdırılıyor
    System.out.println(" Sonuç : ");
    System.out.println(sonuc);

    System.out.println("\nHesaplanma Süresi " + String.format("%.2f", sure) + " milisaniye.");
}
}

```

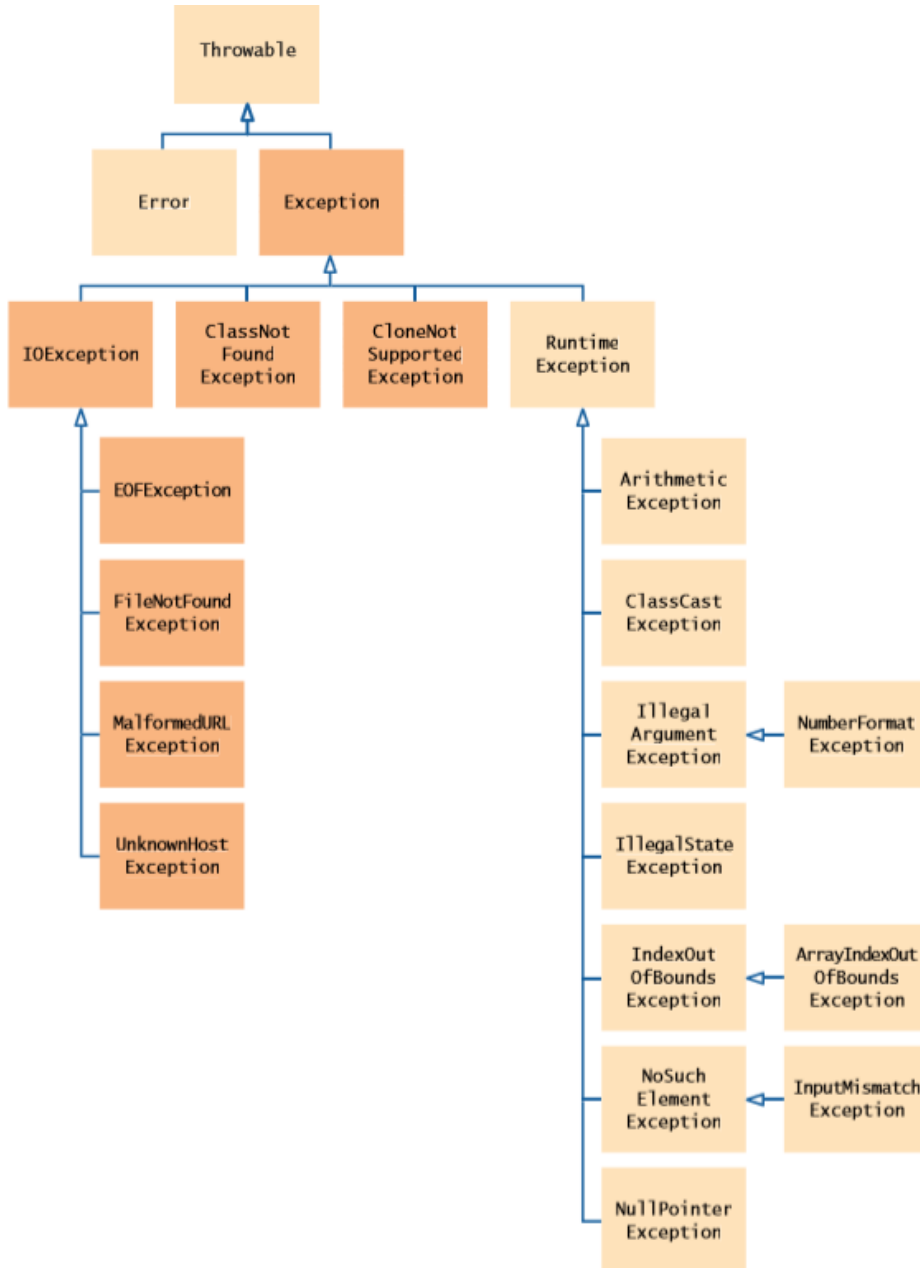
Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK

Programlama Dili Prensipleri

Lab Notları – 12

Hata Yakalama

Programlama dillerinde sıra dışı durumlar ile ilgilenme yöntemi 2 türdür. Raporlama (reporting), Kurtarma (recovery). Hata yakalamadaki en büyük problem, raporlamanın, kurtarmadan çok farklı bir yerde olmasıdır. Örneğin bir liste yapısında Find (bul) metodu aranan elemanı bulma noktasında raporlama olarak elemanın olmadığını bildirebilir. Ama bu noktada ne yapacağını bilemez yani kurtarma kısmı muallaktır. Java programlama dili, hatanın raporlanması ile kurtarma arasında esnek bir mekanizma sağlar. Yapılan işlem, bir hata durumu tespit edildiğinde sıra dışı durumu temsil eden bir nesne fırlatılır. Bu nesnenin hangi sınıftan türetilmesi gerektiğini bilmek için Java'nın hata sınıfları diyagramını bilmek gerekir. Bu diyagram aşağıda görülmektedir.



Örneğin aşağıda bir Hesap sınıfı ve ParaÇek metodu görülmektedir. Bu metod parametre olarak aldığı miktarı hesaptan çeker. Peki ya hesaptaki paradan daha büyük bir miktar çekilmek istenirse ne olacaktır. Burada if kontrolü koyup şartları sağlıyorsa parayı çek demek yeterli olamamaktadır. Çünkü bu sınıfı kullanan kişi bir tepki beklemektedir. Bu metodun birşey döndürmediğini de düşünürsek olumsuz tepki vermek pek mümkün görünmemektedir. Fakat aşağıdaki gibi hatanın fırlatılması en doğru programlama tasarımıdır.

```
public class Hesap {
    private double para;
    public Hesap(){
        para=0;
    }
    public void ParaYatir(double miktar){
        para += miktar;
    }
    public void ParaCek(double miktar){
        if(miktar > para){
            throw new IllegalArgumentException("Yeterli bakiye yok");
        }
        para = para + miktar;
    }
}
```

Yukarıdaki kod bloğunda Hesap sınıfını yazan kişi görevini yerine getirmiş ve istenmeyen bir durumun oluşmasında hatayı fırlatmıştır. Hatanın fırlatılması durumunda miktarın para eklendiği satır çalıştırılmayacak ve hatanın fırlatıldığı noktadan çıkış yapılacaktır. Fakat bu sınıfı kullanacak kişi eğer sadece aşağıdaki gibi kodu kullanırsa hata fırlatılacak fakat yakalanmadığı için program çökecektir.

```
public static void main(String[] args) {
    // TODO code application logic here
    Hesap h = new Hesap();
    h.ParaCek(100);
}
```

Exception in thread "main" java.lang.IllegalArgumentException: Yeterli bakiye yok
at aaab.Hesap.ParaCek(Hesap.java:23)
at aaab.AAAB.main(AAAB.java:21)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)

Aslında yukarıdaki durum kontrol edilmeyen hata fırlatmasından kaynaklanmaktadır. Java'da istenmeyen durumlar iki kategoride incelenir. **Kontrol edilen (checked)** ve **Kontrol edilmeyen (unchecked) durumlar**. Eğer bir hata fırlatma kontrol edilen kategoride ise derleyici programcıyı try, catch blokları içinde kullanmaya zorlar. Bu hata fırlatmasını görmezden gelemezsiniz. Yukarıdaki hata çeşidi kontrol edilmeyen olduğu için derleyici programcıyı try, catch blokları için zorlamaz ve eğer try catch'e konulmaz ve hata durumu oluşursa program çökecektir. Kontrol edilen durumlar genelde dosya işlemleri gibi hayati önem taşıyabilecek durumlarda vardır.

Yazmış olduğunuz hata fırlatan kendi metodlarınızda aşağıdaki gibi yazmanız. Kullanacak programcıyı bilgilendirmek açısından önemlidir.

```
public void ParaCek(double miktar) throws IllegalArgumentException {
    ...
}
```

Önemli: Java'da Eğer metodunuz kontrol edilen bir hata fırlatıyorsa yukarıdaki gibi yazmak zorundasınız.

C tarafına bakıldığında, C programlama dilinde hata fırlatma ve yakalama deyimleri desteklenmez fakat buna benzer yapıyı gerçekleştirmek için jumper yapıları sunar bunlar setjmp.h kütüphanesi içerisinde bulunur. Temel mantık istenmeyen bir durumu önceden sezip programcının oraya jumper değerini değiştirecek bir kod

koymasıdır. Dolayısıyla fonksiyon çalışacağı sırada değer kontrol edilirse istisnai durumun oluşması engellenip programın sonlanmasına izin verilmez.

```
#include "stdio.h"
#include "setjmp.h"
jmp_buf jumper;
int Bol(int x,int y){
    if(y == 0)
        longjmp(jumper, -3);
    return x/y;
}
int main(){
    int a=10, b=0;
    if(setjmp(jumper) == 0){
        printf("%d",Bol(a,b));
    }
    else{
        printf("Sifira Bolunme Hatasi");
    }
    return 0;
}
```

try-catch Blokları JAVA

Programın çökmesini engellemek için sadece hatanın fırlatılması yeterli değildir. Aynı zamanda hatanın yakalanması gerekir. Bu durumda devreye try-catch blokları girer. try bloğu normal yapılmak istenen işlemlerin yazıldığı blok iken catch bloğu hata oluşması durumunda içine girilecek olan bloktur. Bir try bloğu içinde farklı hata sınıfları içeriyorsa farklı sayıda catch bloğu kullanılabilir. Aşağıdaki örneği inceleyelim.

```
...
public void ParaYatir(double miktar) throws ArithmeticException{
    if(miktar <= 0){
        throw new ArithmeticException("Çekilecek miktar sıfırdan büyük olmalıdır.");
    }
    para += miktar;
}
....
```

```
public static void main(String[] args) {
    // TODO code application logic here
    Hesap h = new Hesap();
    try{
        h.ParaCek(100);
        h.ParaYatir(-1);
    }
    catch(IllegalArgumentException ex){
        System.out.println(ex.getMessage());
    }
    catch(ArithmeticException ex){
        System.out.println(ex.getMessage());
    }
}
```

// Yada tek catch bloğunda birleştirilebilir.

```
public static void main(String[] args) {
    // TODO code application logic here
    Hesap h = new Hesap();
    try{
        h.ParaCek(100);
        h.ParaYatir(-1);
    }
```

```

    }
    catch(IllegalArgumentException | ArithmeticException ex){
        System.out.println(ex.getMessage());
    }
}

```

Java'daki aynı kodun C dilinde gerçekleşmesi

```

#ifndef DOSYA_H
#define DOSYA_H

#include "stdio.h"
#include "stdlib.h"
#include "setjmp.h"

struct DOSYA{
    char *yol;
    char *icerik;
    jmp_buf jumper;
    char* (*Oku)(struct DOSYA*);
    void (*Yoket)(struct DOSYA*);
};

typedef struct DOSYA* Dosya;

Dosya DosyaOlustur(char*);
char* TumIcerikOku(const Dosya);
void DosyaYoket(Dosya);
#endif

#include "Dosya.h"

Dosya DosyaOlustur(char *yol){
    Dosya this;
    this = (Dosya)malloc(sizeof(struct DOSYA));
    this->yol=yol;
    this->icerik=NULL;
    this->Oku = &TumIcerikOku;
    this->Yoket=&DosyaYoket;
    return this;
}

char* TumIcerikOku(const Dosya d){
    char *icerik = NULL;
    int boyut = 0;
    FILE *fp;

    fp = fopen(d->yol, "r");
    if(!fp)longjmp(d->jumper,-3);
    fseek(fp, 0, SEEK_END);
    boyut = ftell(fp);
    rewind(fp);

    icerik = (char*) malloc(sizeof(char) * boyut);
    fread(icerik, 1, boyut, fp);
    fclose(fp);

    d->icerik = icerik;
    return icerik;
}

void DosyaYoket(Dosya d){

```

```

        if(d == NULL)return;
        if(d->icerik != NULL)free(d->icerik);
        free(d);
        d=NULL;
    }
#include "Dosya.h"

int main(int argc, char *argv[]){
    Dosya d;
    d=DosyaOlustur("D:\\dene.txt");
    if(setjmp(d->jumper)==0){
        printf("%s",d->Oku(d));
    }
    else printf("Dosya okuma hatasi\n");
    d->Yoket(d);
    return 0;
}

```

Önemli: Bir hata oluşma durumunda ona varsayılan bir değer atamak yerine hatanın fırlatılması en doğru yoldur. Örneğin dosyadan bir sayı okuma örneğinde eğer dosyada sayı yoksa değere sıfır vermek hatalı işlemlere neden olabileceği için doğru değildir. Orada hatanın fırlatılması gerekir.

finally Bloğu

Bazı durumlarda (hatanın oluşsun ya da oluşmasın) özel bir işlem yapmak istenebilir. Böyle bir durumda finally bloğu kullanılmalıdır. Hata oluşsun ya da oluşmasın bu blok çalışacaktır. Bu bloğa şöyle bir senaryoda ihtiyaç duyulabilir. try içerisinde fırlatılan hata catch içerisinde yakalanıp işleyiş devam ederken catch içerisinde tekrar istenmeyen durum oluşmuş ve hata fırlatılmıştır.

Java'da Bazı durumlarda (hatanın oluşması ya da oluşmaması durumlarında) özel bir işlem yapmak isteyebilirsiniz. Böyle bir durumda finally bloğu kullanılmalıdır. Hata oluşsun ya da oluşmasın bu blok çalışacaktır. Aşağıdaki kodu inceleyelim. Burada bakiye ekrana yazdırılmak isteniyor fakat catch bloğu içerisinde tekrar hata fırlatıldığı için yazdırılamıyor.

```

public class Hesap {
    private double para;
    public Hesap(){
        para=0;
    }
    public void ParaYatir(double miktar){
        if(miktar <= 0){
            throw new ArithmeticException("Çekilecek miktar sıfırdan büyük olmalıdır.");
        }
        para += miktar;
    }
    public void ParaCek(double miktar){
        if(miktar > para){
            throw new IllegalArgumentException("Yeterli bakiye yok");
        }
        para = para + miktar;
    }
    @Override
    public String toString() {
        return String.valueOf(para);
    }
}

public static void main(String[] args) {
    Hesap h = new Hesap();
}

```

```

try{
    try{
        h.ParaCek(100);
    }
    catch(IllegalArgumentException ex){
        System.out.println(ex.getMessage());
        h.ParaYatir(-1);
    }
    System.out.println(h);
}
catch(ArithmeticException ex){
    System.out.println(ex.getMessage());
}
}

```

Eğer bakiyenin yazdırıldığı (koyu renkli satır) finally bloğu içerisine alınsaydı hata fırlatılsın ya da fırlatılmasın ekrana yazdırılacaktı.

```

...
finally{
    System.out.println(h);
}
...

```

Önemli: finally bloğunun kullanıldığı kod örneklerine bakıldığında daha çok dosya kapama veri tabanı bağlantısı kesme gibi işlemlerde kullanıldığı görülür. C++'ta finally blokları yoktur. Buradaki amaç sorumluluğun kullanıcıdan tasarımcıya verilmiş olmasıdır. Java programlama dilinde otomatik çöp toplayıcı sistemi vardır dolayısıyla açılmış olan bir kaynak boşa düştüğünde ne zaman çöp toplayıcı tarafından kapatılacağı bilinemez. Burada kod olarak bunu kapatabilecek bir yapıya ihtiyaç vardır bu da finally bloğudur. C++'ta böyle bir sistem olmadığı için finally bloğuna da ihtiyaç yoktur.

Kendi Exception (Hata) Sınıfını Tasarlamak JAVA

Bazen Java'nın sağladığı hata sınıfları sizin hatanızı ifade etmede yetersiz kalabilir. Bu durumda kendi hata sınıfınızı yazıp bu hata sınıfından bir nesne fırlatmanız gerekecektir. Yine yukarıdaki örnek ile devam edersek, YetersizBakiye isminde bir hata sınıfı yazılabilir. Bu hata sınıfını kontrol edilen ya da edilmeyen olarak tasarlayabilirsiniz. Burada karar verilmesi gereken yazacağınız sınıfın hangi Java hata sınıfından kalıtım alacağıdır. Aşağıdaki örneği inceleyelim. Örneğin aşağıda hata sınıfımız kontrol edilen bir hata sınıfı olan IOException sınıfından kalıtım almaktadır. Bu durumda bu hatayı fırlatan her metot try bloğu içerisinde kullanılmalıdır.

```

public class YetersizBakiye extends IOException {
    public YetersizBakiye() { }
    public YetersizBakiye(String hataMesaji){
        super(hataMesaji);
    }
}

...

public void ParaCek(double miktar) throws YetersizBakiye{
    if(miktar > para){
        throw new YetersizBakiye("Yeterli bakiye yok");
    }
    para = para + miktar;
}

...

public static void main(String[] args) {
    // TODO code application logic here
    Hesap h = new Hesap();
}

```

```

try{
    h.ParaCek(100);
}
catch(YetersizBakiye ex){
    System.out.println(ex.getMessage());
}
}

```

Burada kontrol edilen ya da edilmeyen sınıflardan kalıtım alırken dikkat edilmesi gerekmektedir. Çünkü Hesap sınıfını kullanacak programcı her ParaCek metodunu bir try bloğu içerisine yazmak zorunda kalacağı için sıkıntı yaşayabilir. Bu durumda bu tip dosya ile ilgili olmayan hataları kontrol edilmeyen hata sınıflarından kalıtım almak daha doğrudur. Zaten böyle bir durumu önlem alınmaz ise program çökecektir.

Kendi Exception (Hata) Sınıf Tasarımının C dilinde Benzetimi

```

// Hata.h dosyası
#ifndef HATA_H
#define HATA_H

#include "stdio.h"
#include "stdlib.h"
#include "setjmp.h"

struct HATA{
    char* mesaj;
    void (*Yoket)(struct HATA*);
};
typedef struct HATA* Hata;
Hata HataOlustur(char*);
void HataYoket(Hata);

#endif

//Hata.c dosyası
#include "Hata.h"

Hata HataOlustur(char* msg){
    Hata this;
    this=(Hata)malloc(sizeof(struct HATA));
    this->mesaj=msg;
    this->Yoket=&HataYoket;
}

void HataYoket(Hata h){
    if(h == NULL)return;
    free(h);
    h=NULL;
}

// SifiraBolunme.h dosyası
#ifndef SIFIRABOLUNME_H
#define SIFIRABOLUNME_H

#include "Hata.h"
typedef enum BOOL{false, true}boolean;
struct SIFIRABOLUNME{
    Hata super;

```

| |
|---|
| <pre> char* (*Mesaj)(struct SIFIRABOLUNME*); void (*Yoket)(struct SIFIRABOLUNME*); }; typedef struct SIFIRABOLUNME* SifiraBolunme; SifiraBolunme SifiraBolunmeOlustur(); char* HataMesaj(const SifiraBolunme); void SifiraBolunmeYoket(SifiraBolunme); #endif </pre> |
| <pre> // SifiraBolunme.c dosyası #include "SifiraBolunme.h" SifiraBolunme SifiraBolunmeOlustur(){ SifiraBolunme this; this=(SifiraBolunme)malloc(sizeof(struct SIFIRABOLUNME)); this->super = HataOlustur("Sifira Bolunme Hatasi"); this->Mesaj = &HataMesaj; this->Yoket = &SifiraBolunmeYoket; } char* HataMesaj(const SifiraBolunme s){ return s->super->mesaj; } void SifiraBolunmeYoket(SifiraBolunme s){ if(s == NULL)return; s->super->Yoket(s->super); free(s); s=NULL; } </pre> |
| <pre> // Test.c dosyası #include "SifiraBolunme.h" jmp_buf jumper; SifiraBolunme sb=NULL; int Bol(int x,int y){ if(y == 0){ sb = SifiraBolunmeOlustur(); longjmp(jumper, -3); } return x/y; } int main(){ int a=10, b=0; if(setjmp(jumper) == 0){ printf("%d",Bol(a,b)); } else{ printf(sb->Mesaj(sb)); } return 0; } </pre> |
| <p>hepsi: derle calistir</p> <p>derle:</p> <pre>gcc -I ./include/ -o ./lib/Hata.o -c ./src/Hata.c</pre> |

```
gcc -I ./include/ -o ./lib/SifiraBolunme.o -c ./src/SifiraBolunme.c  
gcc -I ./include/ -o ./bin/Test ./lib/Hata.o ./lib/SifiraBolunme.o ./src/Test.c
```

calistir:

```
./bin/Test
```

Hazırlayan
Arş. Gör. Dr. M. Fatih ADAK