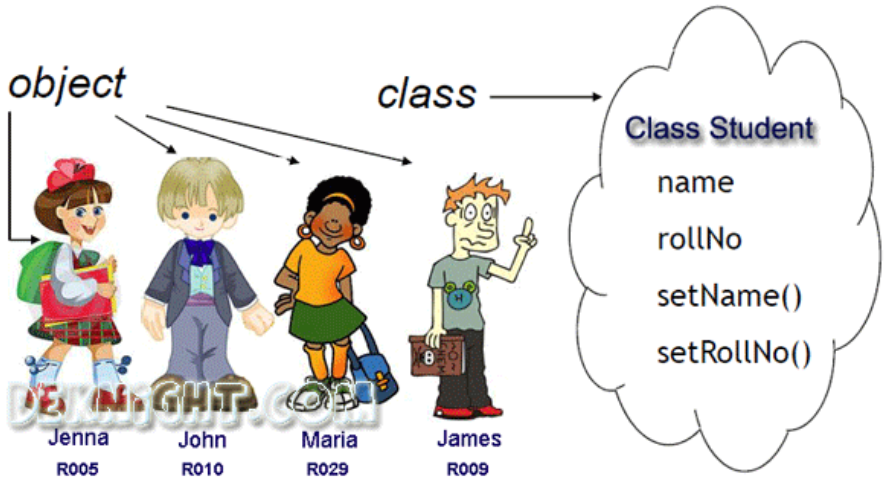
Programlama Dillerinin Prensipleri

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_adı{
        niteleyici(public, private vb.) tür(int, double vb.) degisken_adi;
        niteleyici(public, private vb.) dönüş_türü(int, double vb.) metot_adı(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.

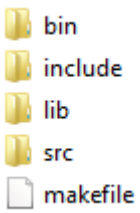
```
public class Kisi {
   private String isim;
   private int yas; // Yıl olarak
   private float boy; // cm
```

```
    private float kilo; // kg
    public Kisi(String ad,float by,float kl){
        isim=ad;
        yas=0;
        boy=by;
        kilo=kl;
    }
    public void YasIlerle(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 içerinde 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 YasIlerle(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 YasIlerle(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 görecek KisiOlustur ve KisiYoket metotları tanımlanmıştır. Bu yapıyı test eden kod parçası aşağıda verilmiştir.

```
#include "Kisi.h"

int main(){
        Kisi k = KisiOlustur("Ahmet",25,5);
        YasIlerle(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.YasIlerle(15);
   }
```

Yukarıda görüldüğü gibi Kişi sınıfından türetilen k nesnesi yıkılması için finalize metodu çağrılmıştır. Ama daha sonra Yasllerle metodu çağrılmış 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 metot 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