Sınıfların Veri Elemanları

Bildirimleri sınıf bildiriminin içerisinde yapılan değişkenlere sınıfın veri elemanları denir. Metotlar ve veri elemanlarına sınıfın elemanları (members) denir. Sınıfın elemanları erişim belirleyicisi alabilir ya da almayabilir. Static olabilir ya da olmayabilir. Örneğin:

```
class Sample {

	int a;
	private double b;
	public static int c, d;

	public static void foo()
	{

	}
}
```

Burada; b, c ve d veri elemanları erişim belirleyicilerini almıştır. a veri elemanı erişim belirleyici anahtar sözcük almamıştır. Bu da farklı bir erişim belirleyicidir. Bu konu daha sonra ele alınacaktır.

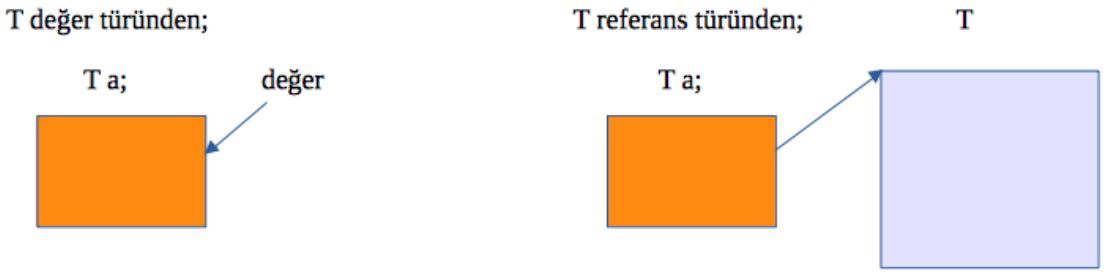
Bir sınıf bildirildiğinde aynı zamanda yeni bir tür de bildirilmiş olur. Temel türlerde olduğu gibi sınıf türünden değişkenler de bildirilebilir. Örneğin:

```
Sample s;
```

Burada s değişkeni Sample türündendir.

Değer Türleri ve Referans Türleri

Java'da bir tür kategori olarak ya değer ya da referans türlerine ilişkindir. T bir tür olmak üzere, T türünden bir değişken eğer doğrudan değerin kendisini tutuyorsa T türü kategori olarak değer türlerine ilişkindir. Eğer T türünden değişken bir adres bilgisi tutuyorsa ve asıl değer o adreste bulunuyorsa T türü kategori olarak referans türlerine ilişkindir.



Bugüne kadar gördüğümüz int, long, double gibi temel türler kategori olarak değer türlerine ilişkindir.

Java'da bütün sınıf türleri kategori olarak referans türlerine ilişkindir. Yani bir sınıf türünden değişken bir adres tutar.

Bir sınıf türünden değişken bildirimi yapıldığında sadece adres tutacak olan bir referans bildirimi yapılmış olur. Ayrıca sınıf nesnesinin kendisini yaratıp adresini bu referansa yerleştirmek (atamak) gerekmektedir. Java'da sınıf nesnelerini yaratmak için new operatörü kullanılmaktadır. new operatörünün genel biçimi şöyledir:

```
new <sınıf ismi>([argüman listesi])
```

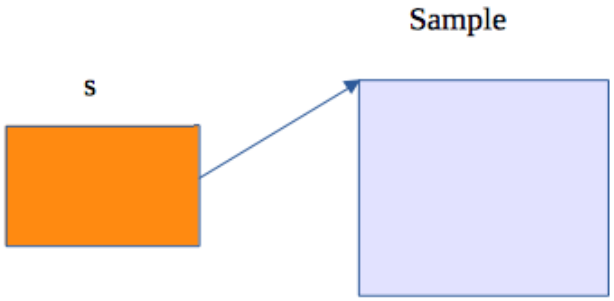
new operatörü nesneyi tahsis eder ve onun adresini üretir. new operatörü ile verilen adres aynı türden bir referansa atanmalıdır. Örneğin:

```
Sample s;

s = new Sample();
```

İşlem referansa ilk değer verilerek de yapılabilir:

```
Sample s = new Sample();
```

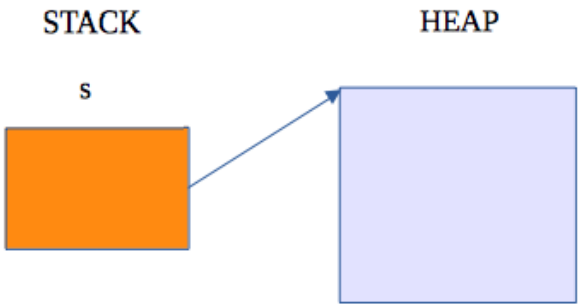


Bilindiği gibi bütün yerel değişkenler ve metotların parametre değişkenleri belleğin stack bölümünde yaratılmaktadır. new operatörüyle yaratılan nesneler belleğin heap denilen bölümünde yaratılırlar.

```
Sample s;

s = new Sample();
```

Burada s yerel değişkeni stack'te bulunacaktır. Ancak s'nin gösterdiği yerdeki nesne heap'tedir.



new operatörü ile tahsis edilen alana sınıf nesnesi (class object) denilmektedir. Sınıf türünden değişkenlere de kısaca referans denilmektedir.

Bir sınıf nesnesi için bellekte (heap) o sınıfın static olmayan veri elemanlarının toplam uzunluğu kadar yer ayrılmaktadır. Sınıfın static veri elemanları ve metotları new ile tahsis edilen alanda yer kaplamaz. Sınıfın static olmayan veri elemanları ardışıl bir blok oluşturur. new operatörü bu bloğun başlangıç adresini verir.

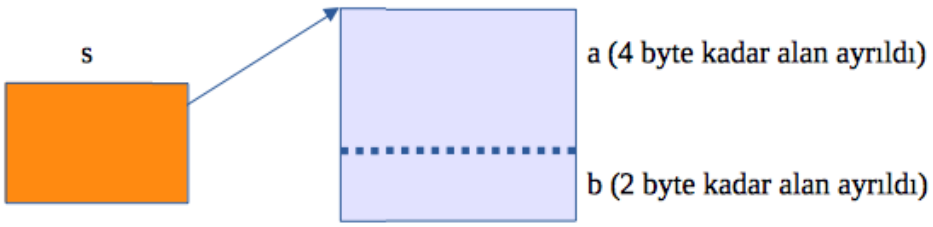
Örneğin:

```
class Sample {
	public int a;
	public short b;
```

```
	//...
}

Sample s;

s = new Sample();
```



Sınıf nesneleri birden fazla parçadan oluşan bileşik nesnelerdir.

Sınıfın metotları belleğin code bölümündedir. Sınıfın static veri elemanları ileride ele alınacaktır.

Sınıfın Veri Elemanlarına Erişim ve Nokta Operatörü

r bir sınıf türünden referans a da bu sınıfın static olmayan bir veri elemanı olmak üzere r.a ifadesi ile r referansının gösterdiği yerdeki nesnenin a parçasına erişilir. Nokta operatörü iki operandlı araek durumunda bir operatördür. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		Sample s;

		s = new Sample();

		s.a = 23;
		s.b = 34;

		System.out.printf("a = %d\nb = %d\n", s.a, s.b);
	}
}

class Sample {
	public int a;
	public short b;
	//...
}
```

Burada, Sample türünden s referansına heap'te yaratılan Sample türünden nesnenin adresi atanmıştır. s.a=23; ifadesi ile s referansının gösterdiği nesnenin a parçasına erişilip değerine 23 verilmiştir. s.b= 34; ifadesi ile de b parçasına erişilip 34 değeri verilmiştir.

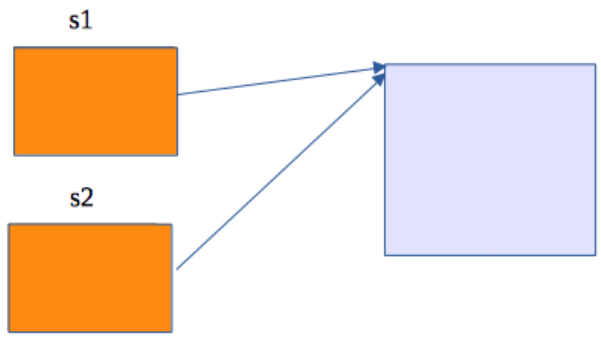
Aynı Türden Referansların Birbirine Atanması

Aynı türden iki referans birbirine atanabilir. Bu durumda referanslar içerisindeki adres değeri aynı olur. Yani bu bir adres atamasıdır. Böylece artık her iki referans da aynı nesneyi gösterir hale gelir. Örneğin:

```
Sample s1, s2;

s1 = new Sample();

s2 = s1;
```



Burada artık s1.a ifadesi ile s2.a ifadesi aynı değişkeni belirtir. Yani aynı nesneyi iki ayrı referans göstermektedir. Artık nesneye hangi referansla erişileceğinin önemi yoktur. Örneğin:

```
package csd;

class App {
   public static void main(String[] args)
  {
        Sample s1, s2;

        s1 = new Sample();

        s1.a = 23;
        s1.b = 34;

        s2 = s1;

        System.out.printf("s1.a = %d\ns1.b = %d\n", s1.a, s1.b); //s1.a = 23 s1.b = 34
        System.out.printf("s2.a = %d\ns2.b = %d\n", s2.a, s2.b); //s1.a = 23 s1.b = 34

        s2.a = 56;

        System.out.printf("s1.a = %d\ns1.b = %d\n", s1.a, s1.b); //s1.a = 56 s1.b = 34
        System.out.printf("s2.a = %d\ns2.b = %d\n", s2.a, s2.b); //s1.a = 56 s1.b = 34
        }
}

class Sample {
        public int a;
        public short b;
        //...
}
```

Farklı türden referanslar birbirlerine doğrudan atanamazlar. Tür dönüştürme operatörü ile de birbirlerine atanamazlar. Örneğin:

```
package csd;
```

```
class App {
	public static void main(String[] args)
	{
		Sample s = new Sample();

		Mample m = (Mample)s; //error:
	}
}

class Sample {
	//...
}

class Mample {
	//...
}
```

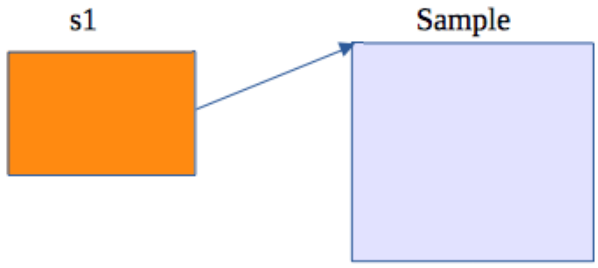
Birden Fazla Nesnenin Yaratılması Durumu

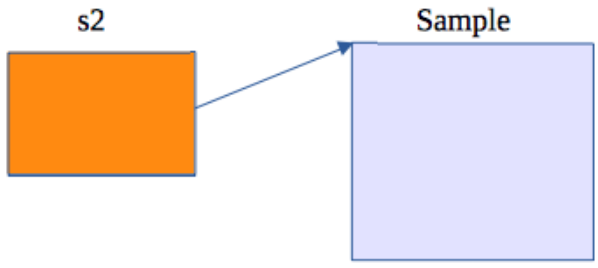
Her new işlemi ile farklı bir nesne yaratılır. Örneğin:

```
Sample s1, s2;

s1 = new Sample();

s2 = new Sample();
```





Artık s1.a ile s2.a aynı değişkenler değildir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		Sample s1, s2;

		s1 = new Sample(); // nesne yaratıldı
		s2 = new Sample(); // nesne yaratıldı
```

```
	}
}

class Sample {
	public int a;
	private static double b;
	protected int c, d;
	float e;

	public static void foo()
	{

	}
}
```

Sınıfın static Veri Elemanları

Sınıfın static veri elemanlarının toplamda tek bir kopyası vardır. Bu elamanlara new işlemi sırasında yer ayrılmaz. Sınıfın bir elemanı ilk kez kullanıldığında yer ayrılır ve programın sonuna kadar bellekte dururlar. Sınıfın static veri elemanlarına sınıf ismi ve nokta operatörü ile erişilir. Örneğin c, Sample sınıfının int türden static bir veri elamanı olmak üzere:

```
Sample.c = 10;
```

ifadesi ile Sample sınıfının static veri elemanına erişilmiştir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		System.out.println(Sample.a);
	}
}

class Sample {

	public static int a = 5;

}
```

Anahtar Notlar: *Aslında Java'da sınıfın static veri elemalarına ya da static metotlarına referans ile de erişmek mümkündür. Bu şekilde erişimin sınıf ismi ile erişmekten bir farkı yoktur. Okunabilirliği/algılanabilirliği düşürdüğü için tercih edilmemelidir. Java'da böyle bir kullanımın faydası yoktur.*

Kavramlar, Nesneler ve Sınıflar

Kavramlar bizim kafamızdadır ve gerçek dünyada yer kaplamazlar. Örneğin, ağaç, doktor, öğrenci gibi kavramlar aslında var olan şeyler değildir. Biz belirli özellikteki nesneleri birbirine benzeterek onlara isimler veririz. Kavramlar aslında insanların karmaşık dünyayı kolay algıyabilmesi için uydurduğu soyut şeylerdir. Nesne yönelimli programlama tekniğinde kavramlar sınıflara karşılık gelir. Bir "Doktor" sınıfı bildirdiğimizde aslında bir doktor oluşturmuş olmayız. Doktorların ortak özelliklerini belirten bir kavram oluşturmuş oluruz. Doktor sınıfı kendi başına bellekte yer kaplamaz. Gerçek hayatta da 'doktor' kavramı fiziksel dünyada yer kaplamaz.

Bir sınıf türünden new işlemi yaptığımızda artık p sınıf türünden gerçek nesne yaratılmış olur. Aynı zamanda yaratılan bu nesne bellekte yer kaplar. Sınıf türünden yaratılan nesnelere İngilizce "instance" denilmektedir. Yani new işlemi ile o kavram türünden nesne yaratılmış olur.

Bir proje nesne yönelimli programlama tekniği ile modellenecekse önce proje içerisindeki kavramlar sınıflar ile temsil edilir. Bu işleme İngilizce "transformation" denilmektedir. Gerçek nesneler sınıf nesneleri biçiminde new operatörüyle yaratılır. Örneğin, bir okul otomasyon sisteminde "Okul, Öğretmen, Öğrenci, Ders, İdari Personel" gibi kavramlar birer sınıfla temsil edilir. Okulda 10 hoca varsa Öğretmen sınıfı türünden 10 nesne new operatörüyle yaratılır. Diğer sınıf türünden de nesneler yaratılarak kodlama işlemine devam edilir. Örneğin tarih kavramı "Date" isimli bir sınıf ile temsil edilebilir.

Bir sınıf bildiriminin kendisi bellekte yer kaplamaz. Sınıf bildirimi derleyiciye new yapıldığı takdirde o nesnenin hangi parçaları olacağı gibi özelliklerini anlatır.

Sınıf Türünden Referans Parametreli Metotlar

Bir metodun parametre değişkeni bir sınıf türünden referans olabilir. Bu durumda bu metot, aynı sınıf türünden bir referans argümanı ile çağrılmalıdır. Böylece nesnenin adresi metoda geçirilmiş olur. Yani bir sınıf nesnesi referans yoluyla metoda aktarılmış olur. Metot içerisinde biz o nesneye erişebiliriz. Örneğin:

```
package csd;

class App {

	public static void foo(Sample s)
	{
		System.out.printf("a=%d%nb=%f%n", s.a, s.b);
	}

	public static void main(String[] args)
	{
		Sample k;

		k = new Sample();

		k.a = 10;
		k.b = 3.14;

		foo(k);
	}
}

class Sample {
	public int a;
	public double b;
	//...
}
```

Geri Dönüş Değeri Bir Sınıf Türünden Olan Metotlar

Bir metodun geri dönüş değeri bir sınıf türünden olabilir. Örneğin:

```
public static Sample foo()

{
	Sample s  = new Sample();
	//…
```

```
	return s;
}
```

Bu durumda o metot çağrıldığında bize o sınıf türünden bir nesnenin adresini (referansını) verir. Örneğin:

```
package csd;

class App {

	public static Sample foo(int a, double b)
	{
		Sample result = new Sample();

		result.a = a;
		result.b = b;

		return result;
	}
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.print("a?");
		int a = Integer.parseInt(kb.nextLine());

		System.out.print("b?");
		double b = Integer.parseInt(kb.nextLine());

		Sample s = foo(a, b);

		System.out.printf("a=%d%nb=%f%n", s.a, s.b);

		kb.close();
	}
}

class Sample {
	public int a;
	public double b;
	//...
}
```

Sınıfların Static Olmayan Metotları

Bir sınıfın static olmayan metotları referans ve nokta operatörüyle çağrılır. Static metotlarda olduğu gibi sınıf ismi ile çağrılamazlar:

Sample s = new Sample();

s.foo(); //foo static olmayan bir metot

Aslında aynı durum veri elemanları için de söz konusudur. Bilindiği gibi sınıfın static olmayan veri elemanları referansla kullanılmaktadır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		Sample s;
```

```
		s = new Sample();

		s.a = 23;
		s.b = 34;

		System.out.printf("a = %d\nb = %d\n", s.a, s.b);
	}
}

class Sample {
	public int a;
	public short b;
	//...
}
```

Sınıfın static olmayan veri elemanları sınıfın static olmayan metotları tarafından doğrudan kullanılabilirler. Ancak sınıfın static olmayan veri elemanları sınıfın static metotları tarafından doğrudan kullanılamazlar. Örneğin:

```
package csd;

class App {

	public static void main(String[] args)
	{
		Sample s;

		s = new Sample();

		s.foo(20);

		System.out.printf("s.a=%d%n", s.a);

	}
}

class Sample {
	public int a;
	public static double b;

	public void foo(int val)
	{
		a = val;
	}

	public static void bar()
	{
		a = 5; // error
	}
}
```

Sınıfın static olmayan metotları içerisinde kullanılan static olmayan veri elemanları metot hangi referansla çağrılmışsa, o referansın gösterdiği yerdeki nesnenin veri elemanlarıdır. Yani örneğin:

```
s.foo();
```

gibi bir çağırma ifadesinde foo'nun içerisinde kullanılan static olmayan veri elemanları aslında s referansının gösterdiği nesnenin veri elemanlarıdır. Örneğin:

```
package csd;

class App {
```

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

		s = new Sample();

		s.foo(20);

		System.out.printf("s.a=%d%n", s.a);
	}
}

class Sample {
	public int a;
	public static double b;

	public void foo(int val)
	{
		a = val;
	}
}
```

Sınıfın static olmayan bir metodu sınıfın başka bir static olmayan metodunu doğrudan çağırabilir. Fakat static bir metodu static olmayan bir metodunu doğrudan çağıramaz. Yani static metotlar içerisinde sınıfın static olmayan veri elemanları kullanılamaz ve sınıfın static olmayan metotları çağrılamaz.

Sınıfın static olmayan bir metodu başka bir static olmayan metodunu çağırdığında çağrılan metot, çağıran metot hangi referans ile çağrılmışsa aynı referansla çağrılmış gibi etki gösterir. Örneğin:

```
package csd;

class App {

	public static void main(String[] args)
	{
		Sample s = new Sample();

		s.foo();
	}
}

class Sample {
	public int a;
	public static double b;

	public void foo()
	{
		System.out.println("foo");
		bar(); // foo yu çağıran referans ile çağrılmış kabul edilir
	}

	public void bar()
	{
		System.out.println("bar");
	}
}
```

Sınfın static metotları referansla çağrılsa bile sınıf ismi le çağrılmış gibi bir etki gösterdiğinden sınıfın static olmayan metotlarını çağıramaz. Eğer çağırabilseydi bu metot içerisinde kullanılan static olmayan veri elemanlarının hangi nesnenin elemanları olduğu anlaşılamazdı. Örneğin:

```
package csd;
```

```
class App {

	public static void main(String[] args)
	{

	}
}

class Sample {

	public static void foo()
	{

		bar(); //error:
		tar();
	}

	public void bar()
	{

	}

	public static void tar()
	{

	}
}
```

Sınıfın static olmayan metotları sınıfın static veri elamanlarını dorudan kullanabilir ve static metotlarını doğrudan çağırabilir. Static elemanlar için referans gereksinimi zaten yoktur. Örneğin:

```
package csd;

class App {

	public static void main(String[] args)
	{

		Sample s = new Sample();

		s.foo();

	}
}

class Sample {
	public int a;
	public static double b;



	public void foo()
	{
		b = 2.23;
		System.out.println("foo");
		bar();
		tar(); //Sample.tar() çağırması ile aynıdır
	}

	public void bar()
	{
		System.out.println("bar");
	}

	public static void tar()
```

```
	{
		System.out.println("tar");
	}
}
```

Sınıfın static bir metodu sınıfın static veri elemanlarını doğrudan kullanabilir ve static metotlarını doğrudan çağırabilir.

Sınıfın elemanı denildiğinde bildirim sınıf içerisinde yapılan metotlar ve veri elemanları anlaşılır. Sınıfın başka elemanları da olabilir. İleride bu konu detaylı anlatılacaktır. Sınfın static elemanları denildiğinde sınıfın static metotları ve static veri elemanları anlaşılır. Sınıfın static olmayan elemanları denildiğide ise static olmayan veri elemanları ve static olmayan metotları anlaşılır. Buna göre yukarıda anlatılan kurallar şöyle özetlenebilir:

1. Sınıfın static olmayan metotları sınıfın hem static elemanlarını hem de static olmayan elemanlarını doğrudan kullanabilir.
2. Sınıfın static metotları yalnızca sınıfın static elemanlarını doğrudan kullanabilir.

Peki bir metodu static yapılıp yapılmayacağına nasıl karar verilir? Çoğu zaman yeni öğrenen programcılar bir metodun ne zaman static ne zaman static olmayan yapılması gerektiğine karar veremezler. Basit bir açıklama şu şekilde yapılabilir:

1. Metot eğer sınıfın static olmayan elemanlarını doğrudan kullanıyorsa zaten static yapılamaz. Mecburen static olmayan bir metot olarak bildirilmelidir.
2. Metot sınıfın hiçbir static olmayan elemanını doğrudan kullanmıyorsa teorik olarak static ya da static olmayan yapılabilir. Bu durumda metodun static yapılması doğru ve iyi bir tekniktir. Çünkü metot static olmayan yapılırsa boşuna o metodu çağırabilmek için bir nesne yaratmak zorunda kalınır.
3. Biz bir sınıfın bir metodunun static olmadığını gördüğümüzde kesinlikle o metodun sınıfın static olmayan bir elemanını kullandığını düşünmeliyiz. Yani “eğer kullanmıyor olsaydı static yapılırdı” şeklinde düşünülebilir.
4. Static olmayan metotlar çağrıldığında doğrudan ya da dolaylı olarak bu çağrı static olmayan bir elemanının kullanılmasına yol açacaktır. Örneğin foo isimli static olmayan bir metot hiçbir static olmayan veri elemanını kullanmıyor olsun fakat bar isimli static olmayan metodunu çağırıyor olsun. Yani aslında bar static olmayan veri elamanlarını kullanıyor durumdadır. Kullanmasaydı bar metodu da static olurdu, bu sebeple foo da static olurdu.