NESNEYE DAYALI PROGRAMLAMA
Dr.Öğr.Üyesi Çiğdem BAKIR

DERS NOTLARI:
C. NESNEYE YÖNELİK İLERİ KAVRAMLAR

BİRE BİR SAHİPLİK İLİŞKİSİ

NESNELER ARASINDAKİ İLİŞKİLER

NESNELER ARASINDAKİ İLİŞKİLER

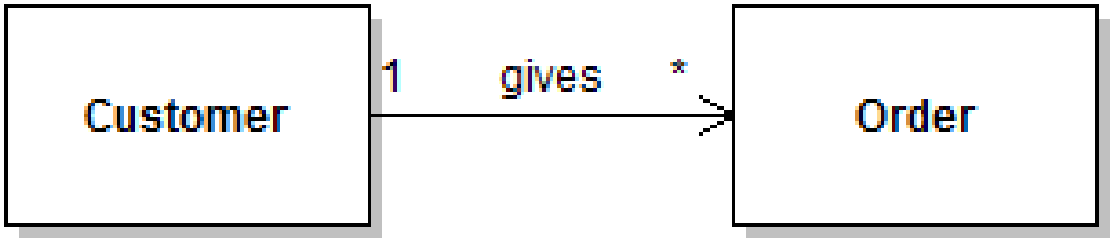
- Bir nesneye yönelik programın, nesneler arasındaki mesaj akışları şeklinde yürüdüğünü gördük.
- Bir nesnenin diğerine bir mesaj gönderebilmesi (yani kullanabilmesi) için, bu iki nesne arasında bir ilişki olmalıdır.
- İlişki çeşitleri:
  - Sahiplik (Association)
  - Kullanma (Dependency)
  - Toplama (Aggregation)
  - Meydana Gelme (Composition)
  - Kalıtım/Miras Alma (Inheritance)
  - Kural koyma (Associative)
- Bu ilişkiler UML sınıf şemalarında gösterilir ancak aslında sınıf örnekleri yani nesneler arasındaki ilişkiler olarak anlaşılmalıdır.

NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

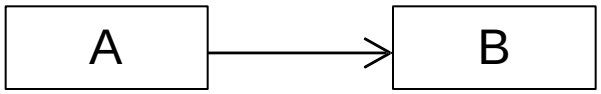
- Bağıntı ilişkisi için anahtar kelime **sahipliktir**.
- Kullanan nesne, kullanılan nesne türünden bir üyeye sahiptir.
- Sadece ilişki kelimesi geçiyorsa, ilişkinin iki nesne arasındaki sahiplik ilişkisi olduğu anlaşılır.
- Bir nesnenin diğerinin yeteneklerini kullanması nasıl olur?
  - Yanıt: Görülebilirlik kuralları çerçevesinde ve metotlar üzerinden.
  - Yani: Mesaj göndererek.

- Örnek: Müşteri ve siparişleri
  - İlişki adları ve nicelikleri de yazılabilir.

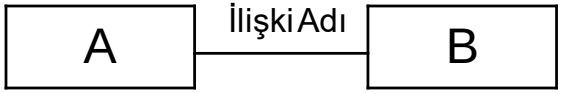


NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)



- Gösterim:

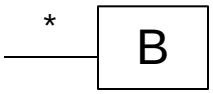


Sahiplik

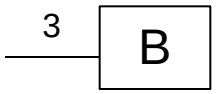
A, B'ye bağımlı = b nesneleri a nesnelerinden habersiz = A kodunda b'nin public metotları çağrılabilir.

Çift yönlü bağımlılık

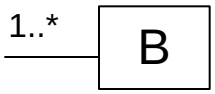
- Okun yönü önemli, kimin kime mesaj gönderebileceğini gösterir.
- Ok yoksa:
  - Ya çift yönlü bağımlılık vardır,
  - Ya da yazılım mimarı henüz bağımlılığın yönünü düşünmemiştir.
- İlişkinin uçlarında sayılar olabilir (cardinality)
  - Çoğulluk ifade eder.
  - İlişkinin o ucunda bulunan nesne sayısını gösterir.



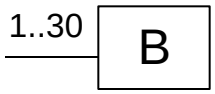
0 veya daha fazla



Tam 3 adet



1 veya daha fazla

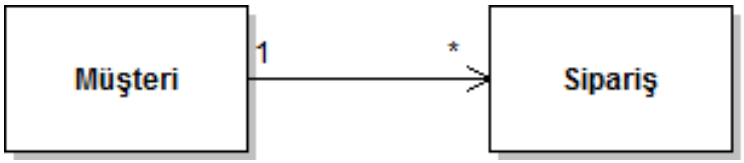


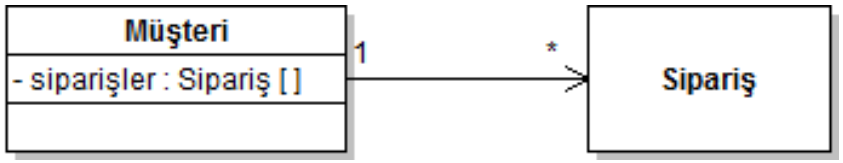
1'den 30'a kadar

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

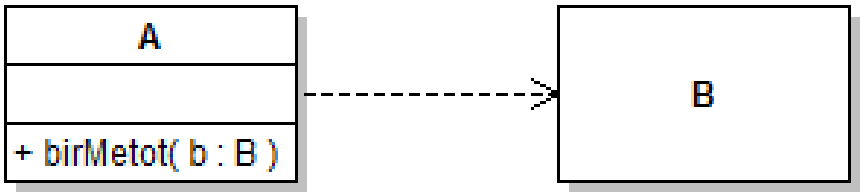
- Sahiplik ilişkisi çizgi ve oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.
  - Ör: Sol alttaki şekil ile sağ alttaki şekil denktir.
  - Diğer ilişkiler için de aynı şey geçerlidir.





KULLANMA İLİŞKİSİ (DEPENDENCY)

- Bir diğerine giden bir mesajın **parametresi** ise veya bir nesne diğerini **sahiplik olmadan kullanıyorsa**.
  - Gösterim:

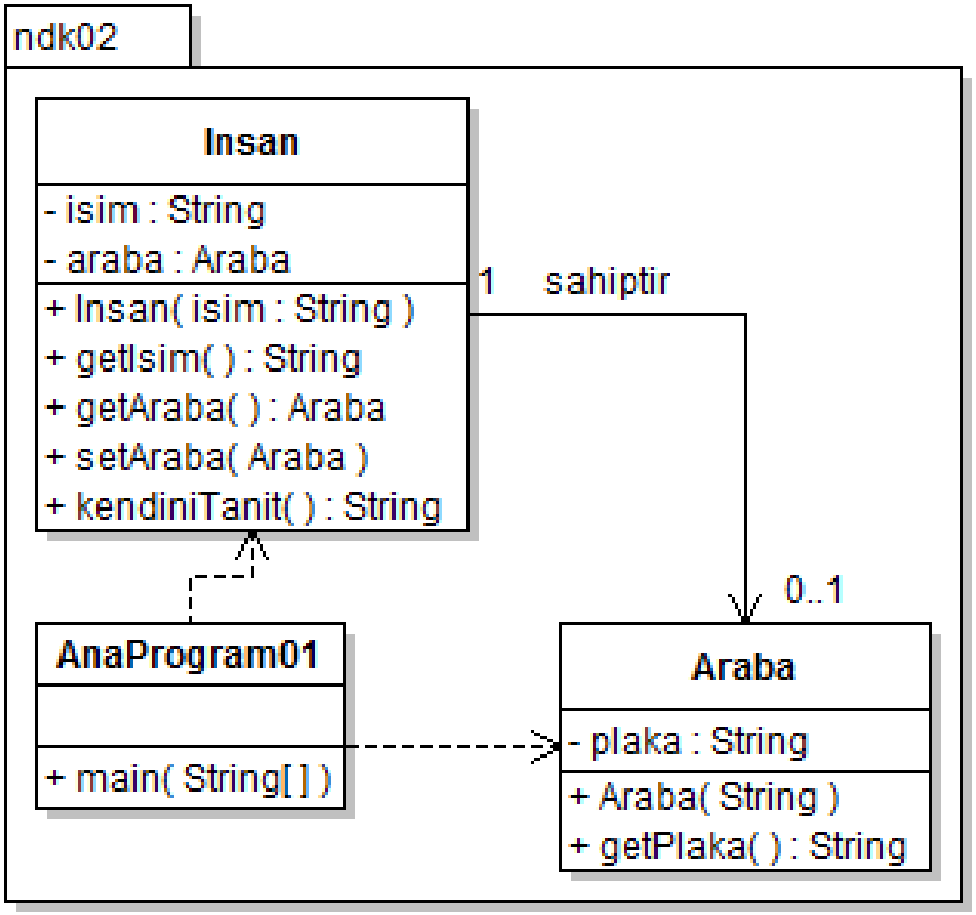


A, B'yi kullanır: A örnekleri birMetot içinden b nesnesine mesaj gönderebilir.

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Her insanın bir arabasının olabileceği bir alan modeli oluşturalım.
- Alan modelini kullanan bir de uygulama yazalım (main metodu içeren).
- Karmaşık yazılımlarda alan modeli ile uygulamanın ayrı paketlerde yer alması daha doğru olacaktır.
- UML sınıf şeması yandadır.
- SORU: Sahiplik ilişkisinin Araba ucu neden 0..1?
- Gizli Bilgi: Araba kurucusuna dikkat



NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Araba sınıfının kaynak kodu:

```
package ndk02;

public class Araba {
    private String plaka;
    public Araba (String plaka) {
       this. plaka = plaka;
    }
    public String getPlaka( ) {
       return plaka;
    }
}
```

- Yukarıdaki koda göre, bir araba nesnesi ilk oluşturulduğunda ona bir plaka atanır ve bu plaka bir daha değiştirilemez.
- Araba sınıfını kodlamak kolaydı, gelelim Insan sınıfına:

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Insan sınıfının kaynak kodu:

```
package ndk02;
public class Insan {
    private String isim;
    private Araba araba;

    public Insan( String isim ) { this.isim = isim; }

    public String getIsim( ) { return isim; }
    public Araba getAraba( ) { return araba; }
    public void setAraba( Araba araba ) {
            this.araba = araba;
    }
    public String kendiniTanit( ) {
            String tanitim;
            tanitim = "Merhaba, benim adım " + getIsim()+ ".";
            if( araba != null )
                    tanitim += "\n" + araba.getPlaka()+" plakalı bir arabam var.";
            return tanitim;
    }
}
```

Dikkat! (Bu da nereden çıktı?)

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- UML sınıf şemamızda Insan - Araba ilişkisinin Araba ucunun 0..1 yazdığı dikkatinizi çekti mi?
  - Bu ne anlama geliyor?
  - Her insanın bir arabası olmayabilir anlamına geliyor.
- Ayrıca:
  - Bir sınıfa bir metot eklenince, hangi metodun hangi sırada çalıştırılacağının, hatta çalıştırılıp çalıştırılmayacağının garantisi yoktur.
    - constructor ve finalizer'ın özel kuralları dışında.
- Buna göre bir insan oluşturulabilir ancak ona araba atanmayabilir.
  - İnsanın arabası olmayınca plakasını nasıl öğrenecek?
  - Bu durumda çalışma anında "NullPointerException" hatası ile karşılaşacaksınız.
  - Ancak bizim sorumluluğumuz, sağlam kod üretmektir. Bu nedenle:
    - İnsanın arabasının olup olmadığını sınayalım, ona göre arabasının plakasına ulaşmaya çalışalım.
      - İnsanın arabası yokken, o üye alanın değeri **`null`** olmaktadır.
      - Yani o üye ilklendirilmemiştir.

NESNE İLİŞKİLERİNİN KODLANMASI

NESNENİN ETKİNLİĞİNİN SINANMASI

- Bir nesne ilklendirildiğinde artık o nesne için etkindir denilebilir.
- nesne1 işaretçisinin gösterdiği nesnenin ilklendirilip ilklendirilmediğinin sınanması:

	İfade	Değer
İlklenmişse (etkinse)	`nesne1 == null`	`false`
	`nesne1 != null`	`true`
İlklenmemişse (etkin değilse)	`nesne1 == null`	`true`
	`nesne1 != null`	`false`

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

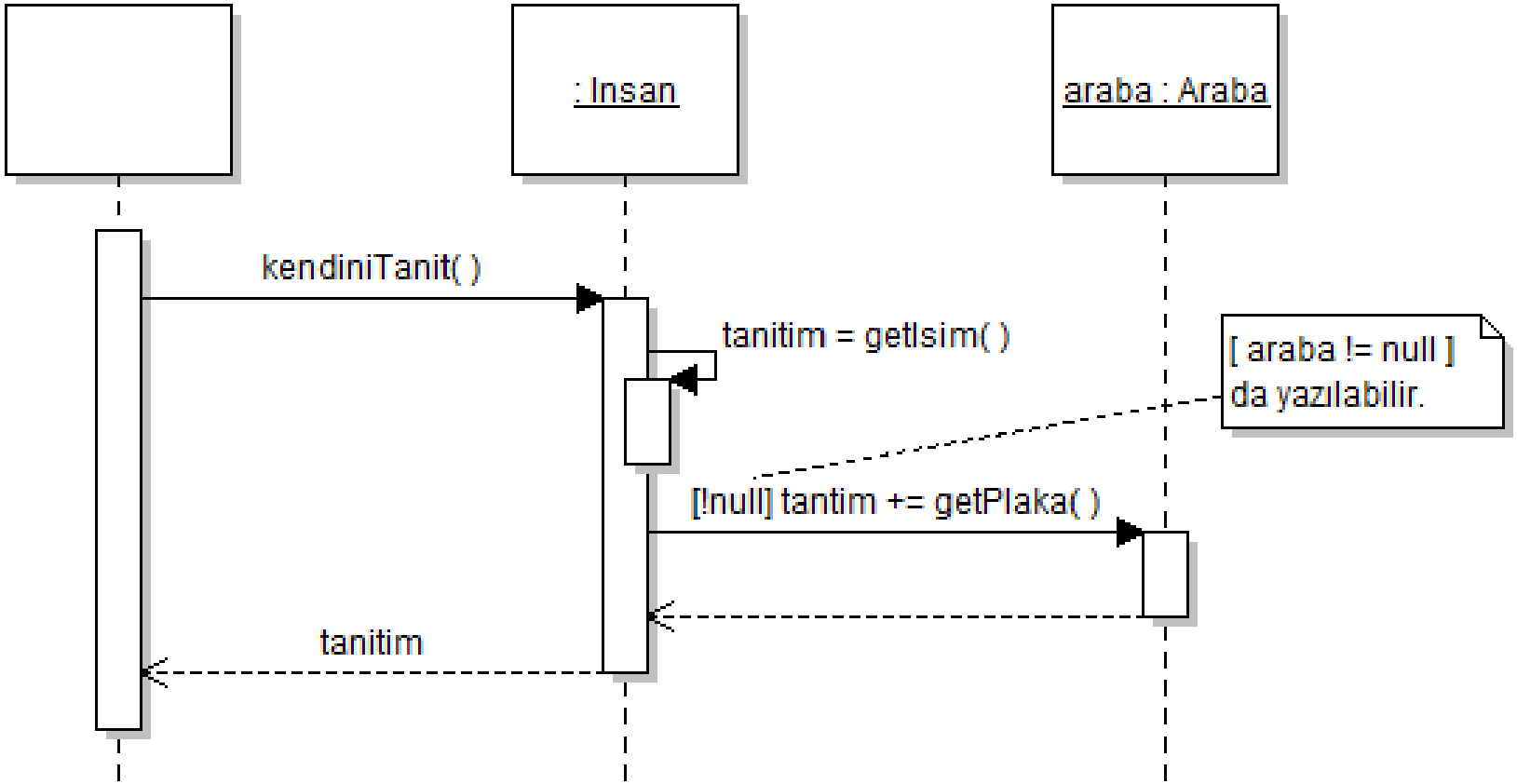
- Nihayet main metodu içeren uygulamamızı yazabiliriz:

```
package ndk02;

public class AnaProgram01{

   public static void main(String[ ] args) {
              Insan oktay;
              oktay = new Insan("Oktay Sinanoğlu");
              Araba rover;
              rover = new Araba("06 RVR 06");
              oktay.setAraba(rover);
              Insan aziz = new Insan("Aziz Sancar");
              System.out.println( oktay.kendiniTanit() );
              System.out.println( aziz.kendiniTanit()  );
   }
}
```

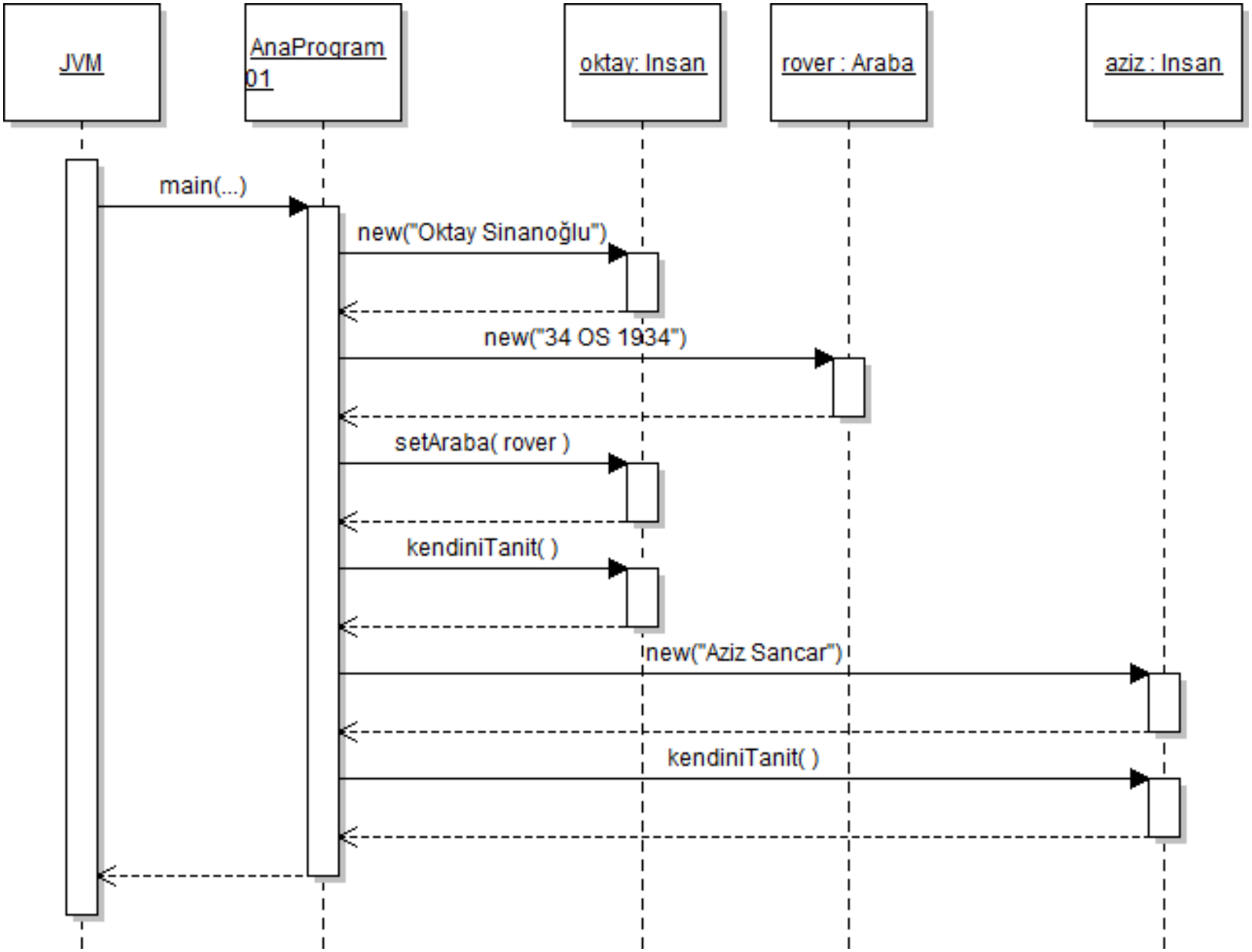
NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Insan sınıfının kendiniTanit metodunun etkileşimşeması:



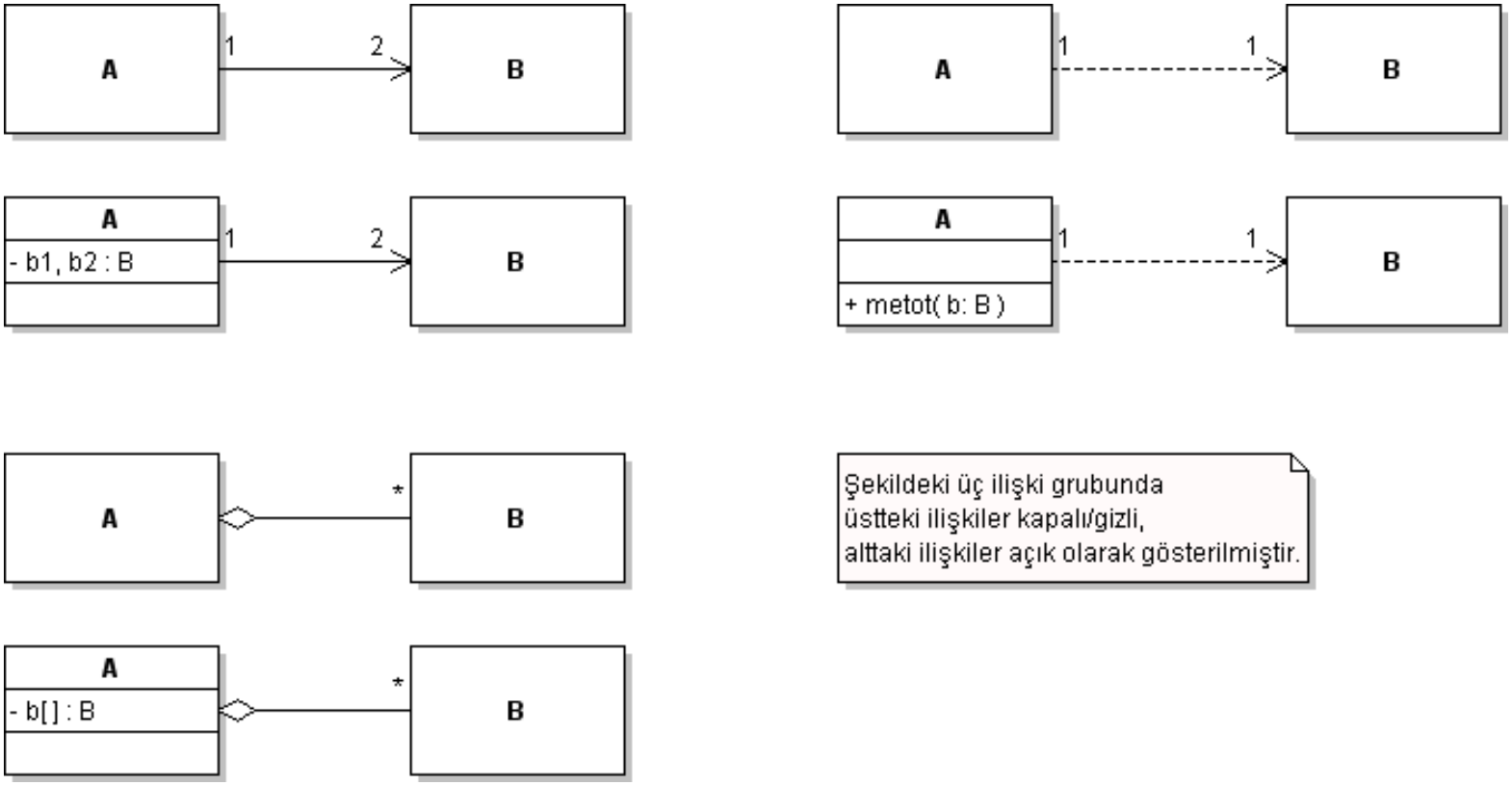
- AnaProgram01 çalıştırılması ile ilgili etkileşim şeması :



NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

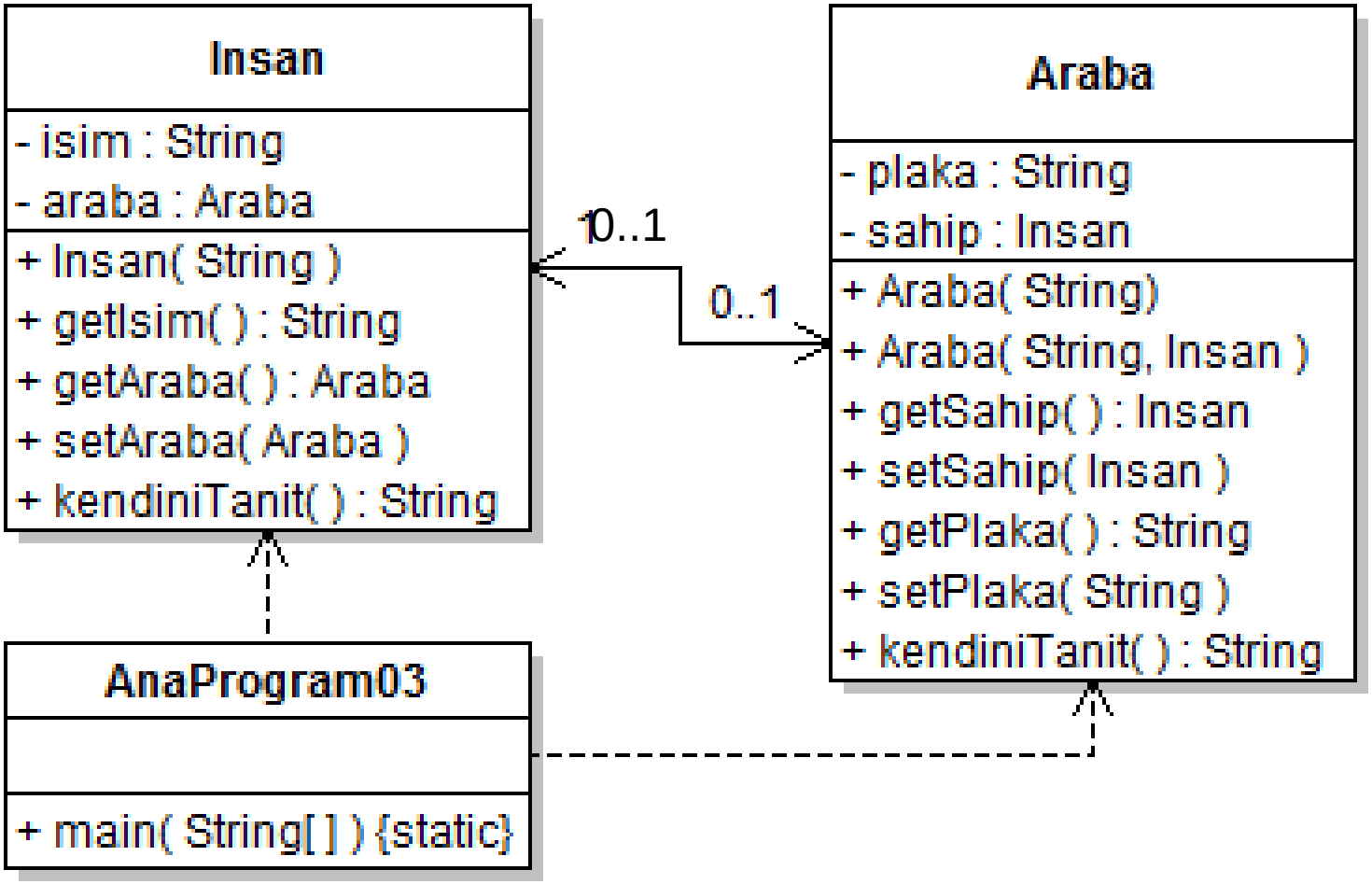
- Sahiplik, kullanma, parça-bütün ilişkileri oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.



NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Hem bir insanın arabası olabilir, hem de bir arabanın kimin arabası olduğunu bilmemiz gerekiyorsa, ilişki çift yönlü kurulmalıdır.



NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Önceki şema ile farkları gördünüz mü?
  - Araba sınıfına sahip üye alanı ve şunlardan en az birini eklemek gerekti:
    - Sahip üyesi için set metodu ve/veya hem plaka hem sahibi alan yapılandırıcı.
    - Sahip üyesi için get metodu
  - Araba sınıfının kaynak kodunu değiştirmemiz gerektiği için paketini de değiştirdik.

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Araba sınıfının yeni kaynak kodu:

```
package ndk03;
public class Araba {
   private String plaka;
   private Insan sahip;
   public Araba( String plakaNo ) { plaka = plakaNo; }
   public Araba(String plaka, Insan sahip) {
            this.plaka = plaka;
            this.sahip = sahip;
   }
   public void setSahip( Insan sahip ) { this.sahip = sahip; }
   public Insan getSahip() { return sahip; }
   public String getPlaka( ) { return plaka; }
   public void setPlaka( String plaka ) { this.plaka = plaka; }
   public String kendiniTanit( ) {
       String tanitim;
       tanitim = "[ARABA] Plakam: " + getPlaka();
       if( sahip != null )
            tanitim += " Sahibimin adı: " + sahip.getIsim();
       return tanitim;
   }
}
```

Attention!

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Neden az önceki kodda if komutuna dikkat çektik?
  - Çünkü birisi Car(String) metodunu çağırıp setOwner metodunu çağırmayı unutabilir.
  - Peki o halde Car(String) kurucusunu silelim mi?
    - Hayır, çünkü gerçek dünyada arabaların fabrikadan çıkar çıkmaz bir sahibi olmaz.

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Yazdıklarımızı denemek için uygulamayı yazalım.

```
01  package ndk03;
02  public class AnaProgram03 {
03      public static void main(String[] args) {
04              Insan oktay = new Insan("Oktay Sinanoğlu");
05              Araba rover = new Araba("06 RVR 06");
06              oktay.setAraba(rover);
07              rover.setSahip(oktay);
08              System.out.println( oktay.kendiniTanit() );
09              System.out.println( rover.kendiniTanit() );
10
11              Insan aziz = new Insan("Aziz Sancar");
12              Araba honda = new Araba("47 AS 1946");
13              aziz.setAraba(honda);
14              honda.setSahip(aziz);
15              System.out.println( aziz.kendiniTanit() );
16              System.out.println( honda.kendiniTanit() );
17      }
18  }
19
20
```

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Önceki uygulamada ne gibi sorunlar görüyorsunuz?
  - Niye hem 6. hem de 7. satırları yazmak zorunda kalalım?
  - Ya o satırları yazmayı unutursak?
  - Ya başka (oktay, rover) – (aziz, honda) ilişkilerini kurarken yanlışlıkla çapraz bağlantı kursak?
  - vb.
- Bu sorunların hepsi, çift yönlü ilişkiyi daha sağlam kurarak ortadan kaldırılabilir.
  - Nereyi değiştirmemiz lazım?

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Insan ve Araba sınıflarının değişen kısımları:

```
package ndk04;
public class Insan {
        /*eski kodu da ekle*/
        public void setAraba(Araba araba) {
                this.araba = araba;
                if( araba.getSahip() != this )
                        this.araba.setSahip(this);
        }
}
```

Dikkat!

```
package ndk04;
public class Araba {
        /*eski kodu da ekle*/
        public void setSahip(Insan sahip) {
                this.sahip = sahip;
                if( sahip.getAraba() != this )
                        this.sahip.setAraba(this);
        }
}
```

Dikkat!

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Main metodu içeren sınıf:

```
package ndk04;
public class Insan {
        /*eski kodu da ekle*/
        public void setAraba(Araba araba) {
                this.araba = araba;
                if( araba.getSahip() != this )
                        this.araba.setSahip(this);
        }
}
```

Dikkat!

```
package ndk04;
public class Araba {
        /*eski kodu da ekle*/
        public void setSahip(Insan sahip) {
                this.sahip = sahip;
                if( sahip.getAraba() != this )
                        this.sahip.setAraba(this);
        }
}
```

Dikkat!

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Bir eksiğimiz daha kaldı, almamız gereken 2. önlem var:
  - Car(String, Insan) kurucumuz da var.

```
package ndk04a;
public class Araba {
   private String plaka;
   private Insan sahip;
   public Araba( String plakaNo ) { plaka = plakaNo; }
   public Araba(String plaka, Insan sahip)  {
            this.plaka = plaka;
            setSahip(sahip);
   }
   public void setSahip( Insan sahip ) { this.sahip = sahip; }
   public Insan getSahip() { return sahip; }
   public String getPlaka( ) { return plaka;  }
   public void setPlaka( String plaka ) { this.plaka = plaka; }
   public String kendiniTanit( ) {
        String tanitim;
        tanitim = "[ARABA] Plakam: " + getPlaka();
        if( sahip != null )
              tanitim += " Sahibimin adı: " + sahip.getIsim();
        return tanitim;
   }
}
```

2. önlem!

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- 2. önlemi almazsak, aşağıdaki gibi bir main ile ilişkinin iki yönü de sağlam kurulmaz:

```
package ndk04a;
public class AnaProgram04 {
        public static void main(String[] args) {
                Insan oktay = new Insan("Oktay Sinanoğlu");
                Araba rover = new Araba("06 OS 1934", oktay);
                System.out.println( oktay.kendiniTanit() );
                System.out.println( rover.kendiniTanit() );
        }
}
```

NESNE İLİŞKİLERİNİN KODLANMASI

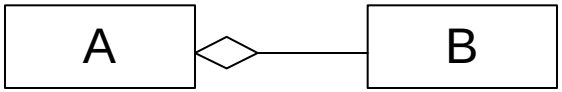
BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Sonuç:
  - Çift yönlü bağıntı tek yönlü bağıntıya göre daha esnektir ancak kodlaması daha zordur.
  - Bu nedenle çift yönlü bağıntıya gerçekten ihtiyacınız yoksa kodlamayın.
  - Peki ya sonradan ihtiyaç duyarsak?
    - Şimdiden kodlamaya çalışıp zaman kaybetmeyin. Zaten yetiştirmeniz gereken bir dolu başka işiniz olacak!

NESNELER ARASINDAKİ İLİŞKİLER

Toplama (Aggregation)

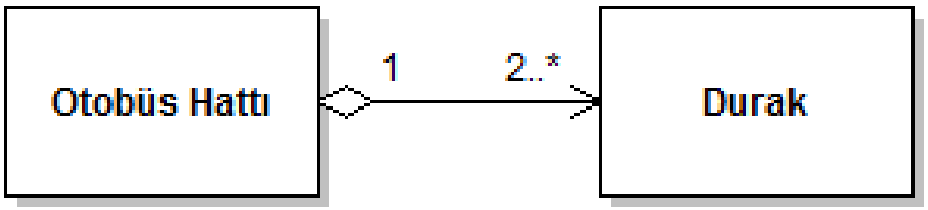
- **Parça-bütün ilişkisini** simgeler.
- Gösterim:



Toplama (aggregation)

- A örneği birden fazla B örneğine sahiptir
- A: Bütün, B: Parça.

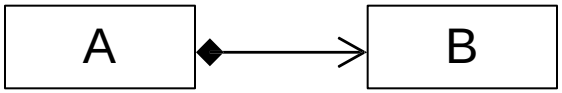
- Şemada gösterilmese de, toplama ilişkisi şunları ifade eder:
  - Elmas ucunda 1 olur
  - Diğer uçta * ve ok olur.
- Toplama, sahiplik ilişkisinden kavramsal olarak daha güçlüdür.
  - Toplama, sıradan sahiplikten daha güçlü kurallara sahiptir.
  - Örneğin, bir otobüs hattı en az iki durağa sahip olmalıdır ve bir hatta yeni duraklar eklemek için uyulması gereken bazı kurallar vardır.



NESNELER ARASINDAKİ İLİŞKİLER

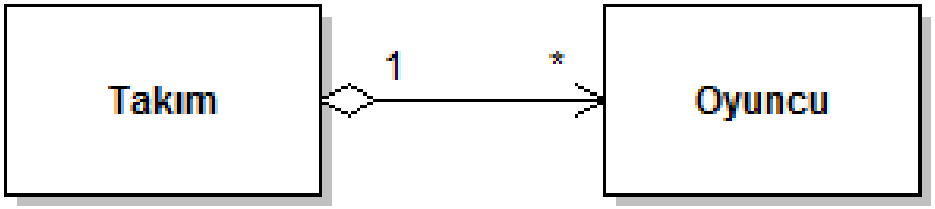
Meydana Gelme (Composition)

- Daha kuvvetli bir parça-bütün ilişkisini simgeler.



Meydana gelme
(composition)

- Meydana gelme ilişkisinde, toplamadan kuvvetli olarak, bir parça aynı anda sadece bir tek bütüne dahil olabilir.
- Örnek:



NESNE İLİŞKİLERİNİN KODLANMASI

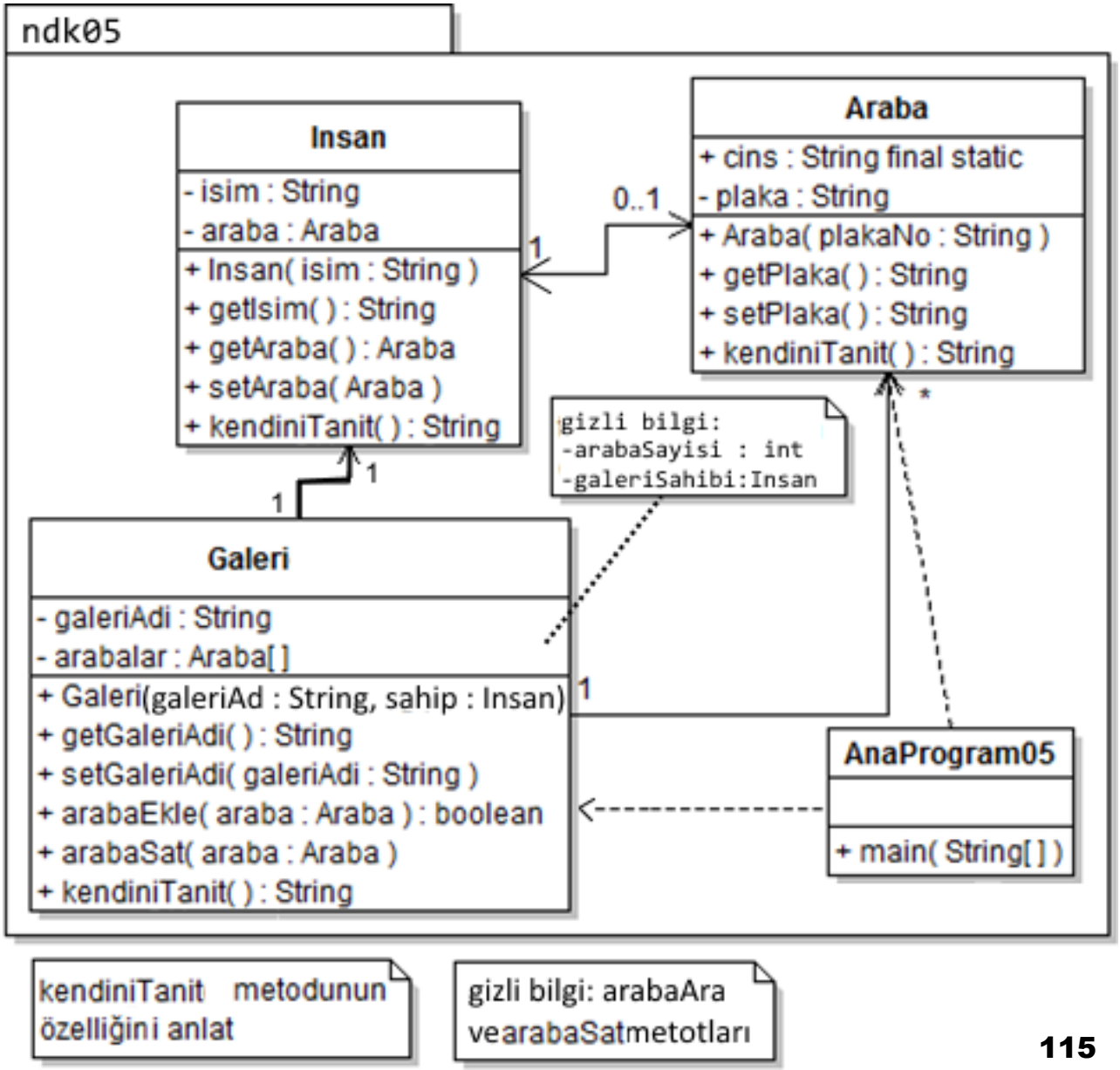
TOPLAMA İLİŞKİSİ

- Örnek: Satılacak birden fazla araba içeren Galeri adlı bir sınıf oluşturalım.
  - Daha önce yazdığımız Araba sınıfını aynen kullanabiliriz.
    - Ek olarak sadece plaka alan bir kurucu yazmak isteyebiliriz. O zaman aynen kullanmamış oluruz.
    - Aynen kullanacaksak:
      - kurucuya sahip üyesini null olarak verebiliriz (NullPointerException?)
      - veya galerinin sahibini verebiliriz (daha emin)
  - Bir Galeri nesnesi birden fazla araba ile ilişkili olabileceğinden toplama veya 1..* sahiplik ilişkisi ile gösterim yapabiliriz.
    - Bu sırada Java'da dizilerin kullanımını ve for döngüsünü de görmüş olacağız.

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- UML sınıf şeması:



NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodu:

```
package ndk05;
public class Galeri {
    private Insan galeriSahibi;
    private String galeriAdi;
    private Araba[ ] arabalar;
    private int arabaSayisi;

    public Galeri( String galeriAd, Insan sahip ) {
                galeriAdi = galeriAd; galeriSahibi = sahip;
                arabaSayisi = 0;
                arabalar = new Araba[30];
    }
    public String getGaleriAdi() { return galeriAdi;  }
    public void setGaleriAdi(String galeriAdi) { this.galeriAdi = galeriAdi; }
    public String kendiniTanit( ) {
                String tanitim;
                tanitim = galeriAdi + " adlı galerinin sahibi: " + galeriSahibi.getIsim();
                tanitim += "\nGaleride " + arabaSayisi + " adet araba  var.";
                return tanitim;
    }
   //devamı var…
```

Not: Burada bir kurucu çalışmadı. Sadece dizi için bellekte yer ayırıldı.

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodunun devamı:

```
public boolean arabaEkle( Araba araba ) {
        if( arabaAra(araba.getPlaka()) != null )
            return false;
        if( arabaSayisi < arabalar.length ) {
            arabalar[ arabaSayisi ] = araba;
            arabaSayisi++;
            return true;
        }
        else
            return false;
}
public Araba arabaAra( String plaka ) {
        for( int i=0; i<arabaSayisi; i++ )
            if( arabalar[i].getPlaka().equalsIgnoreCase(plaka) )
                return arabalar[i];
        return null;
}
//sınıf kodu devam edecek.
```

Burada önce araba zaten eklenmiş mi diye, ayrı bir metot yardımı ile bir denetleme yapmayı akıl edip kodladık.

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodunun devamı:

```
private int arabaBul( String plaka ) {
        for( int i=0; i<arabaSayisi; i++ )
                if( arabalar[i].getPlaka().equalsIgnoreCase(plaka) )
                        return i;
        return -1;
}
public boolean arabaSat( String plaka ) {
        int yer = arabaBul(plaka);
        if( yer != -1 ) {
                for( int i=yer; i<arabaSayisi-1; i++ )
                        arabalar[i] = arabalar[i+1];
                arabaSayisi--; arabalar[arabaSayisi] = null;
                return true;
        }
        return false;
}
} //end class
```

- Alıştırma: arabaSat metodu daha çok arabaSil metodu olmuş. Satışta mali konular devreye girmelidir. Öylesi bir durum nasıl kodlanmalıdır?