SINIFLAR, NESNELER VE ÜYELER

KURUCULAR VE SONLANDIRICILAR

- Kurucu Metot (Constructor):
  - Bir nesne oluşturulacağı zaman sınıfın kurucu adı verilen metodu çalıştırılır.
  - Nesnenin üyelerine ilk değerlerinin atanmasına yarar.
    - Bu yüzden ilklendirici metot olarak da adlandırılırlar.
  - Kurucu metotlara bu derste özel önem gösterilecektir.
- Sonlandırıcı metot:
  - Nesne yok edildiğinde JVM tarafından çalıştırılır.
  - Adı finalize'dır, parametre almaz, geri değer döndürmez.
  - C/C++ aksine, Java programcısının bellek yönetimi ile uğraşmasına gerek yoktur.
    - JVM için ayrılan bellek azalmaya başlamadıkça nesneler yok edilmez.
    - Bu yüzden bir nesnenin finalize metodunu çalıştırmak için çok çabalamanız gerekiyor!
  - Özetle:
    - Bu derste bu konu üzerinde daha fazla durulmayacaktır.

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Kurucu Metot kuralları:
  - Public görünülürlüğe sahip olmalıdır.
  - Kurucu metodun adı, sınıfın adı ile aynı olmalıdır.
  - Bir kurucu metodun geriye o sınıftan bir nesne döndürmesine rağmen,
    - Metot imzasında bir geri dönüş tipi belirtilmez,
    - Metot gövdesinde bir sonuç geri döndürme (return) komutu bulunmaz.
  - Final üyelere değer atamak için uygun bir yerdir.
    - Alternatif: Final üyeye tanımlandığı yerde değer atanması
  - Kod içerisinde bir nesne oluşturulacağı zaman ise, kurucu metot **new** anahtar kelimesi ile birlikte kullanılır.

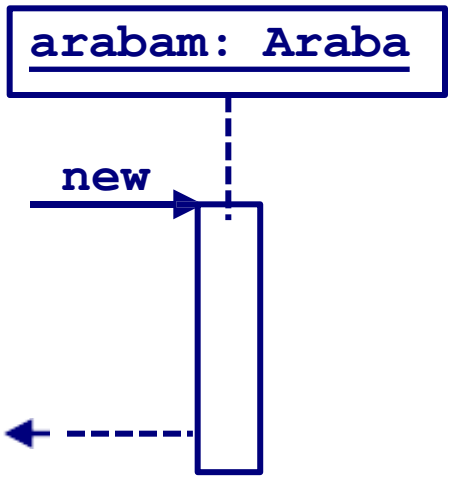
```
arabam = new Araba();
```

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Bir nesneyi kullanmak için onu tanımlamak yetmez, kurucusunu da çalıştırmak suretiyle onu ilklendirmek (*initialize, instantiate*) gerekir.

- UML Gösterimi:



- Kod gösterimi 1: Üye alan olarak kullanım

```
public class AClass {
    private Araba arabam;

    someMethod( ) {
        arabam = new Araba();
    }
}
```

- Kod gösterimi 2: Geçici değişken olarak kullanım

```
public class AnotherClass {
    someMethod( ) {
        Araba arabam = new Araba();
    }
}
```

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Varsayılan kurucu (default constructor):
  - Parametre almayan kurucudur.
  - Programcı tanımlamazsa, JVM (C++: Derleyici) tanımlar.
- Parametreli kurucular:
  - Üye alanlara parametreler ile alınan ilk değerleri atamak için kullanılır.
  - Bir tane bile parametreli kurucu tanımlanırsa, buna rağmen varsayılan kurucu tanımlanmamışsa, varsayılan kurucu kullanılamaz.
- Bir sınıfta birden fazla kurucu olabilir, ancak varsayılan kurucu bir tanedir.
  - Aynı üye aynı sınıf içinde birden fazla tanımlanamaz.
  - Aynı adı paylaşan ancak imzaları farklı olan birden fazla metot tanımlanabilir.
  - Bu tür metotlara **adaş metotlar**, bu yapılan işe ise adaş metot tanımlama (**overloading**) denir.

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

DENETİM AKIŞI

- Denetim akışı: Kodların yürütüldüğü sıra.
  - En alt düzeyde ele alındığı zaman bir bilgisayar programı, çeşitli komutların belli bir sıra ile yürütülmesinden oluşur.
  - Komutların peş peşe çalışması bir nehrin akışına benzetilebilir.
  - Komutların kod içerisinde veriliş sırası ile bu komutların yürütüldüğü sıra aynı olmayabilir.
  - Belli bir komut yürütülmeye başlandığı zaman ise o komut için denetimi ele almış denilebilir.
  - Bu benzetmelerden yola çıkarak, kodların yürütüldüğü sıraya denetim akışı adı verilebilir.

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

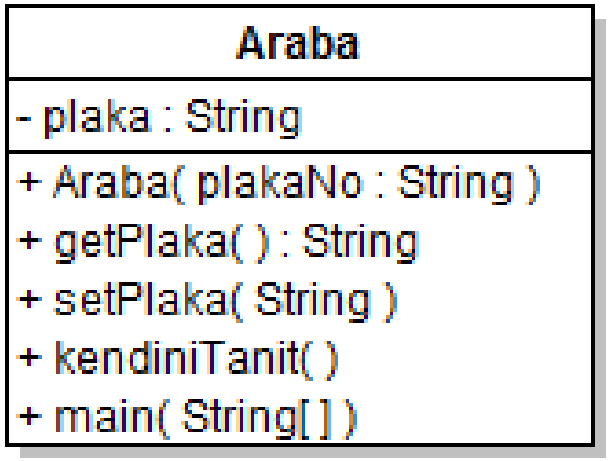
DENETİM AKIŞININ BAŞLANGICI

- Denetim akışının bir başlangıcının olması gereklidir.
  - Akışın hangi sınıftan başlatılacağını programcı belirler.
  - Java'da denetim akışının başlangıcı: Main komutu.
    - public static void main(String[ ] args)
      - static: Henüz bir nesne türetilmedi!
      - args dizisi: Programa komut satırından ilk parametreleri aktarmak için
  - Main metodunun görevi, gerekli ilk bir/birkaç nesneyi oluşturup programın çalışmasını başlatmaktır.
    - Hatırlayın, bir neseneye yönelik programın nesneler arasındaki mesajlar ile yürüdüğünü söylemiştik.
  - Bir sınıfın main metodunun olması, her zaman o metodun çalışacağı anlamına gelmez.
- Blok: Birden fazla komut içeren kod parçası.
  - Kıvrık parantez çifti içerisinde: { ve }

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- UML gösterimi (sınıf şeması)



- Önce UML sınıf şemasını çiz.
- Sonra kodda neresinin şemada nereye denk geldiğini işaretle.
- Pretty printing, camel casing…

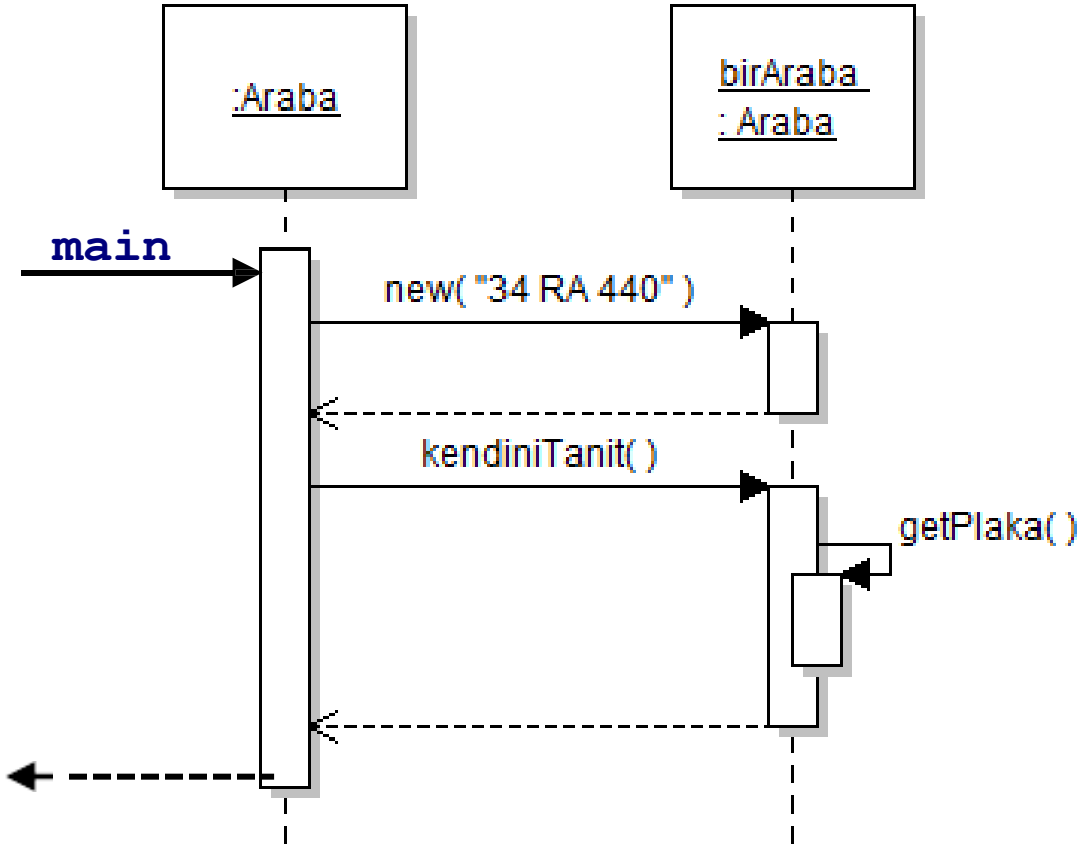
- Kaynak kod (gerçekleme)

```
package ndk01;
public class Araba {
   private String plaka;

   public Araba( String plakaNo ) {
      plaka = plakaNo;
   }
   public String getPlaka( ) {
      return plaka;
   }
   public void setPlaka( String plaka ) {
      this.plaka = plaka;
   }
   public void kendiniTanit( ) {
      System.out.println( "Plakam: " + getPlaka() );
   }

   public static void main( String[] args ) {
      Araba birAraba;
      birAraba = new Araba( "34 RA 440" );
      birAraba.kendiniTanit( );
   }
}
```

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

UML ŞEMASI İLE GÖSTERİM

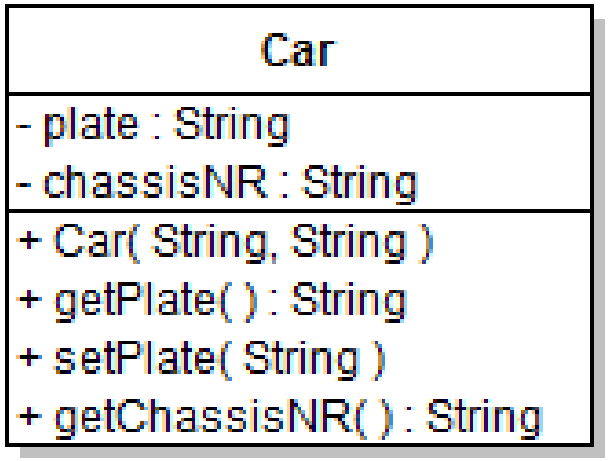
- Örnek koddaki main metodunun, Etkileşim (Interaction) şeması türü olan sıralama şeması (sequence) ile gösterimi.
  - Okların düşeydeki sıralamasına azami dikkat ediniz!



BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- Araba sınıfının başka bir versiyonu:




```
package ndk01;
public class Car {
    private String plate;
    private String chassisNR;
    public Car( String plateNr, String chassisNR ) {
        plate = plateNr;
        this.chassisNR = chassisNR;
    }
    public String getPlate() {
        return plate;
    }
    public void setPlate(String plate) {
        this.plate = plate;
    }
    public String getChassisNR( ) {
        return chassisNR;
    }
}
```

- Araba sınıfının bu versiyonunda bir main metodu yoktur. Bu nedenle doğrudan çalıştırılıp sınanamaz.
- Bu amaçla main metoduna sahip başka bir sınıf kodlamalı ve Car sınıfını oradan test etmeliyiz (ileride gösterilecek).

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- Kurucu metotlara özel önem gösterilmelidir:
  - Gerçek dünyada her aracın bir plakası VE bir şasi numarası bulunur.
  - Bu nedenle iki veri de kurucuda ilklendirilmelidir.
  - Böyle bir kurucunun (en az) iki parametresi olacağı barizdir.
  - Buna göre soldaki kod doğrudur. Sağdaki hem derlemez, hem de hatalıdır (metot imzaları çakışıyor).

```
public class Car {
   private String plate;
   private String chassisNR;
   public Car( String plateNr,
               String chassisNR ) {
      plate = plateNr;
      this.chassisNR = chassisNR;
   }
   /* Rest of the code */
}
```

```
public class Car {
   private String plate;
   private String chassisNR;
   public Car( String plateNr ) {
      plate = plateNr;
   }
   public Car(String chassisNR ) {
      this.chassisNR = chassisNR;
   }
   /* Rest of the code */
}
```

- Hata türleri:
  - Derleme hatası: Kod derleme aşamasında hata verir (derlenmez). Bu nedenle hiç çalıştırılamaz bile.
  - Bug: Kod derler ve çalışır, ancak hatalı sonuçlar üretir, yanlış davranır, vb.
  - Gerçek hayatta bir aracın şasi numarası asla değişmeyeceğinden, sınıfın bu üyesi için bir setter metodu kodlamak da bir bug olacaktır.