Nesne Yönelimli Programlama

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Not: Bu dersin sunumları, "Java Programlama Dili ve Yazılım Tasarımı, Altuğ B. Altıntaş, Papatya Yayıncılık, 2016" kitabı kullanılarak hazırlanmıştır.

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- `this` Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- `finalize()` ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Nesne Yapılandırıcıları

- **Nesneler** kullanıma sunulmadan önce **bazı bilgilere veya bazı işlemlerin yapılmasına gereksinim duyabilir.**
- Uygulama programlarının çalışması sırasında **nesnelerin doğru biçimde başlangıç durumlarına getirilmesi** ve **temizlik işleminin doğru yapılması** gereklidir.
- **Temizlik işleminin doğru yapılmaması durumunda**, daha önceden oluşturulmuş ve artık **kullanılmayan nesneler sistem kaynaklarında gereksiz yer kaplarlar.**
- Başlangıç durumuna getirme işlemleri **yapılandırıcı (constructor)** tarafından gerçekleştirilir.
- **Java** yapılandırıcıyı, **ilgili nesneyi oluşturmadan** hemen **önce otomatik olarak çağırır.**
- **Yapılandırıcı** ile **sınıf isimleri** birebir **aynı olmalıdır.**

3

Nesne Yapılandırıcıları

```
package NesneOrnekler;

class KahveFincani {
    public KahveFincani() {
        System.out.println("Kahve Fincani nesnesi başlatıldı.");
}
}

public class NesneOrnekler {
    public static void main(String[] args) {
        KahveFincani kahvefincani;
        for(int i = 0; i < 5; i++)
            kahvefincani = new KahveFincani();
    }
}
```

```
Kahve Fincani nesnesi başlatıldı.
Kahve Fincani nesnesi başlatıldı.
Kahve Fincani nesnesi başlatıldı.
Kahve Fincani nesnesi başlatıldı.
Kahve Fincani nesnesi başlatıldı.
```

4

Nesne Yapılandırıcıları

- Yapılandırıcılar parametre alabilir.

```
package NesneOrnekler;

class YeniKahveFincani {
    public YeniKahveFincani(int adet) {
        System.out.println(adet + " adet YeniKahveFincani");
    }
}

public class NesneOrnekler {
    public static void main(String[] args) {
        YeniKahveFincani yenikahvefincani;
        for(int i = 0; i < 5; i++)
            yenikahvefincani = new YeniKahveFincani(i);
    }
}
```

```
0 adet YeniKahveFincani
1 adet YeniKahveFincani
2 adet YeniKahveFincani
3 adet YeniKahveFincani
4 adet YeniKahveFincani
```

5

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- **`this`** Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- **`finalize()`** ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Adaş Yordamlar (Overloaded Methods)

- Bir ismin birçok yordam için kullanılması **(method overloading)** kullanım kolaylığı sağlar.
- Aynı isme sahip yordamların ayırt edilebilmesi için **parametreler türlerinin ve sıralarının farklı olması gereklidir.**
- Aynı isme sahip yordamların ayırt edilmesinde **dönen türün farklı olması ayırt edici değildir.**

Adaş Yordamlar (Overloaded Methods)

```
package NesneOrnekler;

public class NesneOrnekler {

    public void topla(long a, long b) {
        System.out.println("Toplam = " + (a + b));
    }

    public void topla(double a, double b) {
        System.out.println("Toplam = " + (a + b));
    }

    public static void main(String[] args) {
        NesneOrnekler mod1 = new NesneOrnekler();
        mod1.topla(10, 20);
        mod1.topla(7.5, 9.5);
    }
}
```

```
Toplam = 30
Toplam = 17.0
```

Adaş Yordamlar (Overloaded Methods)

```
public class NesneOrnekler {

    public int toplamaYap(int a, int b) {
        int sonuc = a + b;
        System.out.println("sonuc - 1 = " + sonuc);
        return sonuc;
    }

    public void toplamaYap(int a, double b) {
        double sonuc = a + b;
        System.out.println("sonuc - 2 = " + sonuc);
    }

    public double toplamaYap(double a, int b) {
        double sonuc = a + b;
        System.out.println("sonuc - 3 = " + sonuc);
        return sonuc;
    }

    public static void main(String[] args) {
        NesneOrnekler mod2 = new NesneOrnekler();
        mod2.toplamaYap(3, 4);
        mod2.toplamaYap(3, 5.5);
        mod2.toplamaYap(6.8, 4);
    }
}
```

```
sonuc - 1 = 7
sonuc - 2 = 8.5
sonuc - 3 = 10.8
```

9

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- **`this`** Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- **`finalize()`** ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Varsayılan Yapılandırıcılar

- Eğer uygulamada herhangi **bir yapılandırıcı oluşturulmazsa, Java bu işlemi kendiliğinden yapmaktadır.**
- **Varsayılan yapılandırıcılar** aynı zamanda **parametresiz yapılandırıcılar (default constructor veya "no-args" constructor)** olarak da anılmaktadır.
- **Varsayılan yapılandırıcılar içi boş yordamlar olarak düşünülebilir.**

Varsayılan Yapılandırıcılar

- Java varsayılan yapılandırıcıyı kendisi oluşturur.

```
class Kedi {
    int i;
}

public class VarsayilanYapilandirici {
    public static void main(String[] args) {
        Kedi kd = new Kedi(); // Varsayılan yapılandırıcı çağrıldı
    }
}
```

```
class Kedi {
    int i;
    /*
     * varsayılan yapılandırıcı bu yapılandırıcıyı eğer biz koymasaydık
     *
     * Java bizim yerimize zaten koyardı
     */
    public Kedi() {
    }
}

public class VarsayilanYapilandirici {
    public static void main(String[] args) {
        Kedi kd = new Kedi(); // Varsayılan yapılandırıcı çağrıldı
    }
}
```

Varsayılan Yapılandırıcılar

- Programcı bir tane yapılandırıcı oluşturduğunda, Java varsayılan yapılandırıcı oluşturmaz.

```
class Araba {
    int kapi_sayisi;
    int vites_sayisi;
    public Araba(int adet) {
        kapi_sayisi = adet;
    }

    public Araba(int adet, int sayi) {
        kapi_sayisi = adet;
        vites_sayisi = sayi;
    }
}

public class VarsayilanYapilandiriciVersiyon2 {
    public static void main(String[] args) {
        Araba ar = new Araba(); // ! Hata var! Bu satır anlamlı değil; yapılandırıcısı yok
        Araba ar1 = new Araba(2);
        Araba ar2 = new Araba(4, 5);
    }
}
```

13

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- **`this`** Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- **`finalize()`** ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

this Anahtar Sözcüğü

- `this` anahtar sözcüğü, **içinde bulunulan nesneyi gösteren bir referans döndürür.**
- `this` ile **nesnelere ait bileşenlere erişim sağlanabilir.**

```
public class NesneOrnekler {
    int gun, ay, yil;

    public void gunEkle(int gun) {
        this.gun = this.gun + gun;
    }

    public void gunuEkranaBas() {
        System.out.println("Gun = " + gun);
    }
    public static void main(String[] args) {
        NesneOrnekler th = new NesneOrnekler();
        th.gunEkle(2);
        th.gunEkle(3);
        th.gunuEkranaBas();
    }
}
```

```
Gun = 5
```

```
public void gunEkle(int gun) {
    gun = gun + gun;
}
```

```
Gun = 0
```

15

this Anahtar Sözcüğü

```
package NesneOrnekler;

public class NesneOrnekler {
    int toplam_yumurta_sayisi = 0;
    NesneOrnekler sepeteKoy() {
        toplam_yumurta_sayisi++;
        return this;
    }

    void goster() {
        System.out.println("toplam_yumurta_sayisi = "
                + toplam_yumurta_sayisi);
    }

    public static void main(String[] args) {
        NesneOrnekler y = new NesneOrnekler();
        y.sepeteKoy();
        y.sepeteKoy();
        y.sepeteKoy();
        y.goster();
        y.toplam_yumurta_sayisi = 0;
        y.sepeteKoy().sepeteKoy().sepeteKoy().goster();
    }
}
```

```
toplam_yumurta_sayisi = 3
toplam_yumurta_sayisi = 3
```

16

`this` Anahtar Sözcüğü

- `this` ile **başka bir yapılandırıcı çağırılabilir.**
- `this` ilk ifade olmalıdır. Birden fazla yapılandırıcı çağırılamaz.

```
public class NesneOrnekler {
    int sayi;
    String malzeme;
    public NesneOrnekler() {
        this(5);
        // this(5,"sucuklu"); !Hata!-iki this kullanılamaz
        System.out.println("parametresiz yapilandirici");
    }
    public NesneOrnekler(int sayi) {
        this(sayi, "Sucuklu");
        this.sayi = sayi;
        System.out.println("Tost(int sayi) ");
    }
    public NesneOrnekler(int sayi, String malzeme) {
        this.sayi = sayi;
        this.malzeme = malzeme;
        System.out.println("Tost(int sayi ,String malzeme) ");
    }
    public void siparisGoster() {
        // this(5,"Kasarli"); !Hata!-sadece yapılandırıcılarda kullanılır
        System.out.println("Tost sayisi=" + sayi + "malzeme =" + malzeme);
    }
    public static void main(String[] args) {
        NesneOrnekler t = new NesneOrnekler();
        t.siparisGoster();
    }
}
```

```
Tost(int sayi ,String malzeme)
Tost(int sayi)
parametresiz yapilandirici
Tost sayisi=5malzeme =Sucuklu
```

17

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- `this` Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- `finalize()` ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Statik Alanlar

- Sadece **global alanlara statik özelliği verilebilir.**
- **Yerel değişkenlerin statik olma özellikleri yoktur.**
- **Global alanları** tür olarak **iki çeşide ayrılabilir:**
  - **statik olan global alanlar**
  - **nesnelere ait global alanlar**
- **Statik alanlar**, bir sınıfa ait olan alanlardır ve bu sınıfa ait **tüm nesneler için ortak bir bellek alanında bulunurlar.**
- Statik tanımlanmış alanlara **sadece bir kez ilk değerleri atanır.**

19

Statik Alanlar

- Statik değişkeni bir nesne değiştirince tüm nesneleri etkiler.

```
public class NesneOrnekler {
    public static int x; // Statik değişken - tüm nesneler için aynı
    public int y; // Dinamik değişken - her nesne için ayrı
    public static void ekranaBas(NesneOrnekler sd) {
        System.out.println("StatikDegisken.x = " + sd.x +
                ", StatikOlmayanDegisken.y = " + sd.y);
    }

    public static void main(String args[]) {
        NesneOrnekler sd1 = new NesneOrnekler();
        NesneOrnekler sd2 = new NesneOrnekler();
        // x = 10;
        // sd1.x = 10 ; // x = 10 ile ayni etkiyi yapar
        // sd2.x = 10 ; // x = 10 ile ayni etkiyi yapar
        sd1.y = 2;
        sd2.y = 8;
        System.out.print("sd1 için -> ");
        ekranaBas(sd1);
        System.out.print("sd2 için -> ");
        ekranaBas(sd2);
    }
}
```

```
sd1 için -> StatikDegisken.x = 0, StatikOlmayanDegisken.y = 2
sd2 için -> StatikDegisken.x = 0, StatikOlmayanDegisken.y = 8
```

20

Statik Alanlar

- Statik değişkeni bir nesne değiştirince tüm nesneleri etkiler.

```
public class NesneOrnekler {
    public static int x; // Statik değişken - tüm nesneler için aynı
    public int y; // Dinamik değişken - her nesne için ayrı
    public static void ekranaBas(NesneOrnekler sd) {
        System.out.println("StatikDegisken.x = " + sd.x +
                "", StatikOlmayanDegisken.y = " + sd.y);
    }

    public static void main(String args[]) {
        NesneOrnekler sd1 = new NesneOrnekler();
        NesneOrnekler sd2 = new NesneOrnekler();
        // x = 10;
        sd1.x = 10 ; // tum nesnelerde x = 10 oldu
        sd2.x = 20 ; // tum nesnelerde x = 20 oldu
        sd1.y = 2;
        sd2.y = 8;
        System.out.print("sd1 için -> ");
        ekranaBas(sd1);
        System.out.print("sd2 için -> ");
        ekranaBas(sd2);
    }
}
```

```
sd1 için -> StatikDegisken.x = 20, StatikOlmayanDegisken.y = 2
sd2 için -> StatikDegisken.x = 20, StatikOlmayanDegisken.y = 8
```

21

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- **`this`** Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- **`finalize()`** ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Statik Yordamlar

- Statik yordamlar (sınıf yordamları) **nesnelerden bağımsız yordamlardır.**
- Statik bir yordamı çağırmak için **herhangi bir nesne oluşturulması zorunlu değildir.**
- **Statik olmayan yordamlardan** (nesneye ait yordamlar) **statik yordamlar çağırılabilir.**
- **Statik yordamlardan nesne yordamları doğrudan çağrılamaz.**

23

Statik Yordamlar

- **Statik olmayan yordamlardan** (nesneye ait yordamlar) **statik yordamları çağırılabilir.**
- **Statik yordamlardan nesne yordamları doğrudan çağrılamaz.**

```
public class NesneOrnekler {
    public static void hesapla(int a, int b) {
        /* static yordam doğrudan nesneye ait bir yordamı çağıramaz */
        // islemYap(a,b); // !Hata!
    }

    public void islemYap(int a, int b) {
        /* doğru , nesneye ait bir yordam, static bir yordamı çağırabilir */
        hesapla(a, b);
    }
}
```

24

Statik Yordamlar

- Statik üyelere **sınıf adı ile erişim yapılabilir.**

```
class Toplama {
    public static double topla(double a, double b) {
        double sonuc = a + b;
        return sonuc;
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
        if (args.length < 2) {
            System.out.println("Ltf iki adet sayi giriniz");
            System.exit(-1); // uygulama sonlanacaktır
        }
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);
        System.out.println(a +" " + b);
        double sonuc = Toplama.topla(a, b); // dikkat
        System.out.println("Sonuc : " + sonuc);
    }
}
```

```
1.0 2.0
Sonuc : 3.0
```

25

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- **`this`** Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- **`finalize()`** ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

`finalize()` ve Çöp Toplayıcı

- **Yapılandırıcılar** sayesinde **nesneler oluşturulurken değer atanabilmektedir.**
- Oluşturulan **nesneler daha sonradan bellekten silinmektedir.**
- **Java** programlama dilinde **işleri bitince nesneler otomatik olarak hafızadan atılır.**
- Java programlama dilinde, **bir nesnenin gerçekten çöp olup olmadığına karar veren mekanizma çöp toplayıcısıdır (garbage collector).**
- Çöp toplama sistemi, **kodu yazan kişi için büyük bir rahatlık oluşturmaktadır.**

27

`finalize()` ve Çöp Toplayıcı

- Çalışmakta olan uygulamanın içerisinde bulunan **bir nesne artık kullanılmıyorsa,** bu nesne **çöp toplayıcısı tarafından bellekten silinir.**
- Çöp toplayıcı (garbage collector) **bir nesneyi bellekten silmeden hemen önce** o nesnenin **`finalize()` yordamını çağırır.**
- Böylece bellekten silinecek olan nesnenin yapması gereken **son işlemler var ise** bu işlemler **`finalize()` yordamı içerisinde yapılır.**

28

`finalize()` ve Çöp Toplayıcı

Aşağıdaki örnekte **`finalize()`** çağırılmadığından çöp toplayıcısı çalışmadı.

```
class Elma {
    int i = 0;
    Elma(int y) {
        this.i = y;
        System.out.println("Elma Nesnesi Olusturuluyor = " + i);
    }
    public void finalize() {
        System.out.println("Elma Nesnesi Yok Ediliyor = " + i);
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
        for (int y = 0; y < 5; y++) {
            Elma e = new Elma(y);
        }
        for (int y = 5; y < 11; y++) {
            Elma e = new Elma(y);
        }
    }
}
```

```
Elma Nesnesi Olusturuluyor = 0
Elma Nesnesi Olusturuluyor = 1
Elma Nesnesi Olusturuluyor = 2
Elma Nesnesi Olusturuluyor = 3
Elma Nesnesi Olusturuluyor = 4
Elma Nesnesi Olusturuluyor = 5
Elma Nesnesi Olusturuluyor = 6
Elma Nesnesi Olusturuluyor = 7
Elma Nesnesi Olusturuluyor = 8
Elma Nesnesi Olusturuluyor = 9
Elma Nesnesi Olusturuluyor = 10
```

29

`finalize()` ve Çöp Toplayıcı

Aşağıdaki örnekte **`System.gc()`** ile çöp toplayıcısı çağırıldı.

```
class Elma2 {
    int i = 0;
    Elma2(int y) {
        this.i = y;
        System.out.println("Elma2 Nesnesi Olusturuluyor = " + i);
    }
    public void finalize() {
        System.out.println("Elma2 Nesnesi Yok Ediliyor = " + i);
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
        for (int y = 0; y < 10; y++) {
            Elma2 e = new Elma2(y);
        }
        System.gc(); // çöp toplayıcısını çağırdık
        for (int y = 10; y < 21; y++) {
            Elma2 e = new Elma2(y);
        }
    }
}
```

```
Elma2 Nesnesi Olusturuluyor = 0
Elma2 Nesnesi Olusturuluyor = 1
Elma2 Nesnesi Olusturuluyor = 2
Elma2 Nesnesi Olusturuluyor = 3
Elma2 Nesnesi Olusturuluyor = 4
Elma2 Nesnesi Olusturuluyor = 5
Elma2 Nesnesi Olusturuluyor = 6
Elma2 Nesnesi Olusturuluyor = 7
Elma2 Nesnesi Olusturuluyor = 8
Elma2 Nesnesi Olusturuluyor = 9
Elma2 Nesnesi Olusturuluyor = 10
Elma2 Nesnesi Yok Ediliyor = 3
Elma2 Nesnesi Olusturuluyor = 11
Elma2 Nesnesi Yok Ediliyor = 1
Elma2 Nesnesi Yok Ediliyor = 0
Elma2 Nesnesi Olusturuluyor = 12
Elma2 Nesnesi Yok Ediliyor = 9
Elma2 Nesnesi Olusturuluyor = 13
Elma2 Nesnesi Yok Ediliyor = 8
Elma2 Nesnesi Olusturuluyor = 14
Elma2 Nesnesi Yok Ediliyor = 7
Elma2 Nesnesi Olusturuluyor = 15
Elma2 Nesnesi Yok Ediliyor = 6
Elma2 Nesnesi Olusturuluyor = 16
Elma2 Nesnesi Yok Ediliyor = 5
Elma2 Nesnesi Olusturuluyor = 17
Elma2 Nesnesi Yok Ediliyor = 4
Elma2 Nesnesi Olusturuluyor = 18
Elma2 Nesnesi Yok Ediliyor = 2
Elma2 Nesnesi Olusturuluyor = 19
Elma2 Nesnesi Olusturuluyor = 20
```

`finalize()` ve Çöp Toplayıcı

- System sınıfının statik bir yordamı olan **`gc()`, çöp toplayıcısının kodu yazan kişi tarafından tetiklenmesini sağlar.**
- Böylece çöp toplayıcısı, **çöp haline gelmiş olan nesneleri** (kullanılmayan nesneleri) **bularak** (eğer varsa) **bellekten siler.**
- Örnekte; ilk for döngüsünde oluşturulan **Elma2 nesneleri**, **döngü bittiğinde çöp halini alacaklardır.**
- Bunun nedeni, ilk for döngüsünün bitimiyle, oluşturulan 10 adet Elma2 nesnesinin erişilemez bir duruma geleceğidir.
- Doğal olarak, erişilemeyen **bu nesneler**, **çöp toplayıcısı tarafından hafızadan atılmaktadır.**

31

`finalize()` ve Çöp Toplayıcı

Aşağıdaki örnekte referansı olmayan **2. nesne bellekten silinmiştir.**

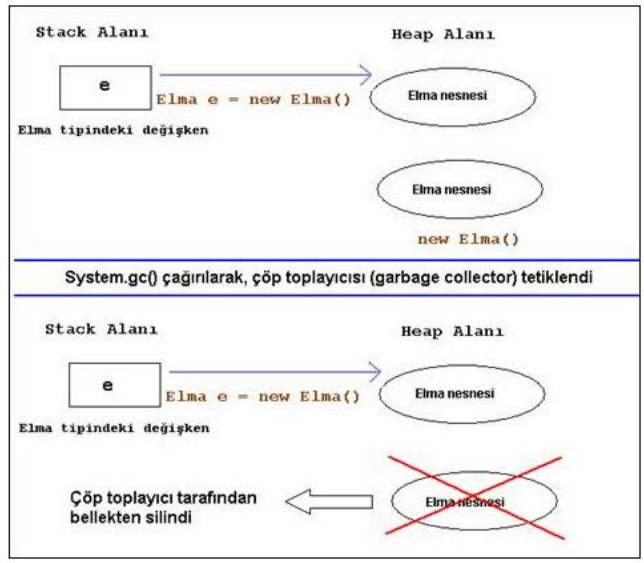
```
class Elma {
    int i = 0;
    Elma(int y) {
        this.i = y;
        System.out.println("Elma Nesnesi Olusturuluyor = " + i);
    }
    public void finalize() {
        System.out.println("Elma2 Nesnesi Yok Ediliyor = " + i);
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
        Elma e = new Elma(1);
        new Elma(2);
        System.gc(); // çöp toplayıcısını çağırdık
    }
}
```

```
Elma Nesnesi Olusturuluyor = 1
Elma Nesnesi Olusturuluyor = 2
Elma2 Nesnesi Yok Ediliyor = 2
```

32

`finalize()` ve Çöp Toplayıcı

Örnekteki referansı olmayan **2. nesnenin bellekten silinişi**



`finalize()` ve Çöp Toplayıcı

Örnekte işlerin bir kısmı **`finalize()`** yordamında yapılmaktadır.

```
class Ucak {
    String ucak_isim;
    boolean benzin_deposu_dolu = false;
    boolean benzin_deposu_kapagi_acik_mi = false;
    Ucak(boolean depoyu_doldur, String ucak_isim) {
        benzin_deposu_kapagi_acik_mi = true;     //kapağı açıyoruz
        benzin_deposu_dolu = depoyu_doldur;      //depo dolu
        this.ucak_isim = ucak_isim;
    }

    /* Kapakların kapatılmasını finalize() yordamına bıraktık */
    public void finalize() {
        if (benzin_deposu_kapagi_acik_mi) {         // kapak açıksa
            benzin_deposu_kapagi_acik_mi = false;   // kapağı kapa
            System.out.println(ucak_isim + "- kapaklari kapatildi ");
        }
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
          Ucak ucak_1 = new Ucak(true,"F-16");  // benzin doldur
          new Ucak(true,"F-14");                // benzin doldur
          System.gc();                          // kapakları kapat
          System.out.println("Ucaklara benzin dolduruldu, kapaklari"
                                                    + "kapatildi");
      }
}
```

```
Ucaklara benzin dolduruldu, kapaklarikapatildi
F-14- kapaklari kapatildi
```

Konular

- Nesne Yapılandırıcıları
- Adaş Yordamlar (Overloaded Methods)
- Varsayılan Yapılandırıcılar
- `this` Anahtar Sözcüğü
- Statik Alanlar
- Statik Yordamlar
- `finalize()` ve Çöp Toplayıcı
- Nesne Değişkenlerine Değer Atanması

Nesne Değişkenlerine Değer Atanması

- **Java** ile geliştirilen **uygulamalarda üç tür değişken çeşidi bulunur:**
  - **yerel (local) değişkenler**
  - **nesneye ait global alanlar**
  - **sınıfa ait global alanlar (statik alanlar)**
- Bu değişkenlerin tipleri **temel (primitive)** veya herhangi bir **sınıf tipinde olabilir.**

Nesne Değişkenlerine Değer Atanması

- Statik fonksiyondan non-statik değişkene erişim yapılamaz.
- Yerel değişken statik olamaz.

```
class StatikOrnek{
    int x = 10;          // nesneye ait global alan
    static int y = 20;   // sınıfa ait global alan

    public void statikornek() {
        System.out.println(x + ", " + y);
    }
}

public class NesneOrnekler {
    public static void main(String args[]) {
        int i;       // yerel degisken
        // static int y = 5 ; // yanlış
        StatikOrnek st = new StatikOrnek();
        st.statikornek();
    }
}
```

```
10, 20
```

```
class StatikOrnek{
    int x = 10;          // nesneye ait global alan
    static int y = 20;   // sınıfa ait global alan

    public static void statikornek() {
        System.out.println(x + ", " + y); // x' e erişim yapılamaz.
    }
}
```

Nesne Değişkenlerine Değer Atanması

- **Nesnelere ait global alanlara** ilk değerleri programcının vermesi zorunlu değildir.
- **Java** bu alanlara **ilk değerleri kendiliğinden verir.**

```
Veri Tipleri Ilk Degerleri
boolean false
char [ ] 0
byte 0
short 0
int 0
long 0
float 0.0
double 0.0
```

```
public class NesneOrnekler {
    boolean mantiksal_deger;
    char krakter_deger;
    byte byter_deger;
    short short_deger;
    int int_deger;
    long long_deger;
    float float_deger;
    double double_deger;
    public void ekranaBas() {
        System.out.println("Veri Tipleri Ilk Degerleri");
        System.out.println("boolean " + mantiksal_deger);
        System.out.println("char [" + krakter_deger + "] " +
                (int) krakter_deger);
        System.out.println("byte " + byter_deger);
        System.out.println("short " + short_deger);
        System.out.println("int " + int_deger);
        System.out.println("long " + long_deger);
        System.out.println("float " + float_deger);
        System.out.println("double " + double_deger);
    }

    public static void main(String args[]) {
        new NesneOrnekler().ekranaBas();
        /*
         * // yukaridaki ifade yerine
         * // asagidaki ifadeyide kullanabilirsiniz.
         * NesneOrnekler nesneornekler = new NesneOrnekler();
         * nesneornekler.ekranaBas();
         *
         */
    }
}
```

Nesne Değişkenlerine Değer Atanması

- Aksi belirtilmediği sürece **nesnelere ait global alanlara**, herhangi bir **sınıf tipinde olması durumunda**, **başlangıç değeri olarak "null" atanır (boş değer).**
- Eğer global alanlar bu içeriğiyle kullanılmaya kalkışılırsa, **çalışma anında (run time) hata ile karşılaşılır.**
- Hata ile karşılaşmamak için ilgili **alanları uygun nesnelere bağlamak gerekir.**

```
public class NesneOrnekler {
    String s;
    public static void main(String args[]) {
        NesneOrnekler nt = new NesneOrnekler();
        System.out.println("s = " + nt.s);
        // nt.s = nt.s.trim(); //hata
    }
}
```

```
s = null
```


Nesne Değişkenlerine Değer Atanması

- Sınıflara ait **global alanlara** (statik alanlar) değer atama ile **nesnelere ait global alanlara** değer **atama aynı şekilde yapılır.**

```
Veri Tipleri Ilk Degerleri
static boolean false
static char [ ] 0
static byte 0
static short 0
static int 0
static long 0
static float 0.0
static double 0.0
```

```
public class NesneOrnekler {
    static boolean mantiksal_deger;
    static char krakter_deger;
    static byte byter_deger;
    static short short_deger;
    static int int_deger;
    static long long_deger;
    static float float_deger;
    static double double_deger;
    public void ekranaBas() {
        System.out.println("Veri Tipleri Ilk Degerleri");
        System.out.println("static boolean " + mantiksal_deger);
        System.out.println("static char [" + krakter_deger + "] " +
                (int) krakter_deger);
        System.out.println("static byte " + byter_deger);
        System.out.println("static short " + short_deger);
        System.out.println("static int " + int_deger);
        System.out.println("static long " + long_deger);
        System.out.println("static float " + float_deger);
        System.out.println("static double " + double_deger);
    }

    public static void main(String args[]) {
        new NesneOrnekler().ekranaBas();
        /*
         * // yukaridaki ifade yerine
         * // asagidaki ifadeyi de kullanabilirsiniz.
         * NesneOrnekler nesneornekler = new NesneOrnekler();
         * nesneornekler.ekranaBas();
         */
    }
}
```

Nesne Değişkenlerine Değer Atanması

- **Nesnelere ait global alanlara başlangıç değerleri hemen verilir** (yapılandırıcılardan (constructor) önce).
- Belirtilen **alanların konumu hangi sırada** ise **başlangıç değeri alma sırasında aynı olur.**
- **Statik alanlar**, sınıflara ait olan alanlardır ve **statik olmayan alanlara** (nesne alanları) göre **başlangıç değerlerini daha önce alırlar.**

41

Nesne Değişkenlerine Değer Atanması

```
class Kagit {
    public Kagit(int i) {
        System.out.println("Kagit (" + i + ") ");
    }
}

public class NesneOrnekler {
    Kagit k1 = new Kagit(1); // dikkat

    public NesneOrnekler() {
        System.out.println("Defter() yapilandirici ");
        k2 = new Kagit(33); // artık başka bir Kagit nesnesine bağlı
    }

    Kagit k2 = new Kagit(2); // dikkat

    public void islemTamam() {
        System.out.println("Islem tamam");
    }

    Kagit k3 = new Kagit(3); // dikkat

    public static void main(String args[]) throws Exception {
        NesneOrnekler d = new NesneOrnekler();
        d.islemTamam();
    }
}
```

```
Kagit (1)
Kagit (2)
Kagit (3)
Defter() yapilandirici
Kagit (33)
Islem tamam
```

42

Nesne Değişkenlerine Değer Atanması

```
public class NesneOrnekler {
    static Catal c1 = new Catal(1, "statik alan"); // dikkat!
    public NesneOrnekler() {
        System.out.println("Kahvalti() yapilandirici");
    }

    Tabak t1 = new Tabak(1, "statik-olmayan alan"); // dikkat!
    public void islemTamam() {
        System.out.println("Islem tamam");
    }

    static Catal c2 = new Catal(2, "statik alan"); // dikkat!
    public static void main(String args[]) throws Exception {
        NesneOrnekler d = new NesneOrnekler();
        d.islemTamam();
    }

    static Tabak t4 = new Tabak(4, "statik alan"); // dikkat!
    static Tabak t5 = new Tabak(5, "statik alan"); // dikkat!
}
```

```
class Peynir {
    public Peynir(int i, String tur) {
        System.out.println("Peynir (" + i + ") -->" + tur);
    }
}

class Tabak {
    public Tabak(int i, String tur) {
        System.out.println("Tabak (" + i + ") -->" + tur);
    }
    static Peynir p1 = new Peynir(1, "statik alan");
    Peynir p2 = new Peynir(2, "statik-olmayan alan");
}

class Catal {
    public Catal(int i, String tur) {
        System.out.println("Catal (" + i + ") --> " + tur);
    }
}
```

```
Catal (1) --> statik alan
Catal (2) --> statik alan
Peynir (1) -->statik alan
Peynir (2) -->statik-olmayan alan
Tabak (4) -->statik alan
Peynir (2) -->statik-olmayan alan
Tabak (5) -->statik alan
Peynir (2) -->statik-olmayan alan
Tabak (1) -->statik-olmayan alan
Kahvalti() yapilandirici
Islem tamam
```

43

Nesne Değişkenlerine Değer Atanması

- Statik alanlara toplu olarak değer atanabilir.

```
class Kopek {
    public Kopek() {
        System.out.println("Hav Hav");
    }
}

public class NesneOrnekler {
    static int x;
    static double y;
    static Kopek kp;
    {
        x = 5;
        y = 6.89;
        kp = new Kopek();
    }

    public static void main(String args[]) {
        new NesneOrnekler();
    }
}
```

```
Hav Hav
```

44

Nesne Değişkenlerine Değer Atanması

- Statik olmayan alanlara toplu olarak değer atanabilir.

```
class Dinozor {
    public Dinozor() {
        System.out.println("Ben Denver");
    }
}

public class NesneOrnekler {
    int x;
    double y;
    Dinozor dz;
    {
        x = 5;
        y = 6.89;
        dz = new Dinozor();
    }

    public static void main(String args[]) {
        new NesneOrnekler();
    }
}
```

```
Ben Denver
```

45