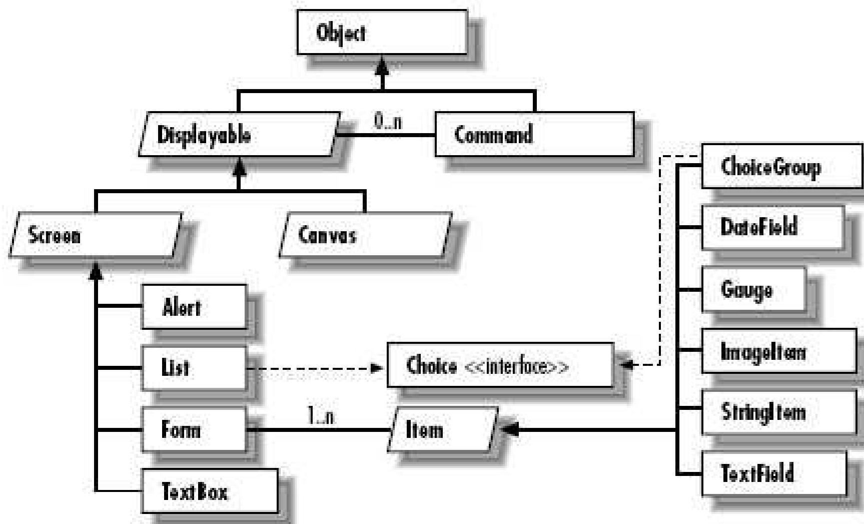
UI (User Interface – Kullanıcı Arayüz) Bileşenleri

J2ME uygulamalarını kullanıcı tarafında sunmak için belli arayüzleri kullanır. Bu her programlama dilindeki gibi kullanıcı arayüzü yani user interface bileşenleridir. Bu bileşenler bize kullanıcıdan veri almak yada veri göndermek için görsel bileşenler oluşturma imkanı verir.

UI bileşenleri iki türdür diyebiliriz bunlar yüksek seviye ve düşük seviye bileşenlerdir. Yüksek seviye bileşenler tamamen çalışan sisteme bağlı ve çalıştığı işletim sisteminin kütüphanelerini kullanan bileşenlerdir. Yani bir uygulama bir cihaz üzerinde çalışırken yüksek seviye bir bileşene sahipse bu bileşen cihaz üzerindeki kütüphaneleri kullanarak oluşturulur. Bir örnek verecek olursak ileriki aşamalarda göreceğimiz TextBox bileşeni yüksek seviye bir bileşendir bu bileşeni kullandığımız bir uygulama hazırladığımızı varsayalım. Çalışma anında uygulama aygıta erişerek o aygıta özgü TextBox kütüphanesini çağırıp ekranda bunu gösterecektir. Dolayısıyla bu görünüm her telefonda ayrı olabilir. Ancak farklılıklar çok büyük olmayıp aslında bize telefonun ekranına ve kullanım koşullarına uygun bir erişim imkanı sağlar. Düşük seviye bileşenler ise görünüm sabittir tüm cihazlarda aynı şekilde görünür. Düşük seviye bileşenlar Canvaslardır. Canvas ekrana grafik çizimleri için kullanılır (yazı, resim çizgi vs...) dolayısıyla bunun cihazlar arasında farklılık göstermesi gibi bir durum söz konusu olamaz. Örnek olarak ekrana çizdiğimiz bir araba resminin ekranlar arası farklılık göstermesi gibi bi durum söz konusu değildir ancak burada ekran boyutlarının farklılığından kaynaklanan bir sorun olabilir bunu ileriki konularda nasıl aşacağımızı göreceğiz.



UI bileşenlerinin yapısı yukarıdaki gibidir. Yüksek seviyeli bileşenlerin tamamı Screen sınıfından türemişlerdir.
Bu bileşenler aşağıdaki gibidir.
Alert
List
TextBox

Form
ChoiceGroup
DataField
Gauge
ImageItem
StringItem
TextField

Düşük seviye bileşenleri ise sadece Canvas oluşturur.

Cep telefonlarında pc platformundaki gibi butonlar kullanılamaz bu sadece bazı dokunmatik ekran özelliğine sahip telefonlarda mümkün olabilir. J2ME ortamında buton yerine Command ları kullanırız. Bu sayede herhangi bir hareket olacağı zaman telefonlarda Command dinleyicisi devreye girer ve commandAction metodunu çalıştırır. Bu metod içerisinde hangi işlem yapıldığını algılayıp ilgili işlemlerimizi gerçekleştirebiliriz.

Display

UI bileşenlerini ekranda göstermemiz için öncelikle ekrana sahip olmamız gerekir bu yüzden bir nesneyi ekrana atamadan önce ekran sahip olmalıyız. Bunun için Display tipinde bir nesne tanımlamalıyız. Display nesnesinin static bir metodu olan getDisplay() metodu ile belirttiğimiz bir midletin ekranını alabiliriz. Bu metoda almamız gereken midlet nesnesinin referansını göndermemiz gerekiyor.

```
Display.getDisplay(MIDlet m);
```

Yukarıdaki kod bloğu ile ekran nesnesine sahip oluruz. Örneklemek gerekirse

```
Display ekran = Display.getDisplay(this);
```

Burada ekran adında Display tipinde bir nesne yarattık. Bu nesneye Display sınıfının getDisplay metodu ile ulaştık ve parametre olarak this kelimesini gönderdik. Burada this kelimesi içerisinde bulunduğumuz MIDlet i belirtir. Yani bulunduğumuz MIDlet e ait bir ekran almış oluyoruz. Aksi bir durumda farklı bir MIDlet in referansınıda gönderebilirdik. Display tipinde bir referansa sahip olduktan sonra bu nesneye belli bir UI bileşenini atayabiliriz. Bu bileşen TextBox, Form, List, Alert ve Canvas olabilir. Aynı anda tek bir bileşen ekranda görülebilir. Yanlız Form nesnesine ait olan bazı bileşenler form nesnesine eklendikten sonra nesne ekrana verilebilir bu durumda yine tek bir nesne ekrana atanmış fakat form nesnesi içerisinde bir çok nesne olmuş olur.

```
Display ekran=Display.getDisplay(this);
ekran.setCurrent(nesne);
```

Yukarıdaki örnek yapıda aldığımız ekrana bir bileşen atıyoruz. Bu nesne yine TextBox, Form, List, Alert yada Canvas olabilir bu yapıyı örneklendirelim:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
public class OrnekMIDlet extends MIDlet {
   public void startApp() {
      TextBox ornek1= new TextBox("Ornek1","Ornek1",20,0);
      TextBox ornek2= new TextBox("Ornek2","Ornek2",20,0);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(ornek1);
      ekran.setCurrent(ornek2);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Örneğimizde iki adet TextBox bir adette Display tipinde değişken yarattık. TextBox larımız ornek1 ve ornek2 dir. Öncelikle ornek1 nesnesini setCurrent() motodu ile ekranımıza atıyoruz artık ornek1 ekranımızda görülecektir. Sonrasında *ekran.setCurrent(ornek2)* ile ornek2 nesnemizi atadık bu durumda artık ekranda sadece ornek2 görünecektir. Yani ikinci adımdaki setCurrent() metodu birincisini ezmiştir.
Bu kodumuzun ekran çıktısı aşağıdaki gibidir.

Göründüğü gibi ekranımızda sadece Ornek2 mevcut. Aslında burada önce Ornek1 görünmüş sonrasında ise Ornek2 ekrana verilerek ekranı kaplamıştır. NetBeans te kodumuzu debug durumunda kodumuzu çalıştırırsak öncelikle Ornek1 in göründüğünü rahatlıkla görebiliriz.

TextBox

J2ME ortamında yazı tipinde veri girişi yapmak için kullandığımız bileşenlerden biri TextBox' tır. Bu nesne tüm ekranımızı kaplayarak bize tam sayfa görünümünde veri girişi yapma imkanı verir. Bu veri girişi harf, sayı veya klavyemizin desteklediği herhangi bir karakter olabilir. TextBox' lar tüm yüksek seviye bileşenler gibi telefonlar arasında farklılıklar gösterebilir.
TextBox yapısı aşağıdaki gibidir.

```
TextBox (String title, String text, int maxSize, int constraints)
```

Bu yapıda istenen dört parametre sırası ile title, text, maxSize ve constraints' tir.

title	Yazı giriş alanının üstünde bulunacak başlıktır.
text	Veri giriş alanımızın içerisinde başlangıçta bulunacak yazı.
maxSize	Veri giriş alanımızın alabileceği karakter sayısı
constraints	Girilecek verinin kontrolü.

Constraints bize belli bir veri giriş tipi sunar örnek olarak 0 gönderirsek istediğimiz herhangi bir karakteri girebiliriz. Ancak kod içerisine sayı girmek kafa karıştırabilir ve çoğu zaman hangi sayının neye karşılık geldiğini unutabiliriz. Bu yüzden sayı girmek yerine bir form bileşeni olan TextField sınıfına ait veri giriş tiplerinin sayı karşılıklarını döndüren static değişkenleri kullanabiliriz. Bu değişkenler bize kullanmamız gereken sayıları daha anlamlı bir şekilde gösterir. Örnek olarak 0 kullanacağımız yerde TextField.ANY diyebiliriz burada dönecek değişken bize herhangi bir karakter girişi yapmamızı sağlayabilir. TextField nesnesine ait bu değişkenler final ve statictir bu yüzden değiştirilemez ve nesnesi oluşturulması gerekmez.
TextField Nesnesine ait kullanabileceğimiz diğer tipler aşağıdaki gibidir.

ANY	Herhangi bir karakter
EMAILADDR	E-mail adresi
NUMERIC	Sayı
PHONENUMBER	Telefon numarası
URL	İnternet adresi
DECIMAL	Ondalıklı
PASSWORD	Şifre

Constraints bize bir çok kolaylıkta sağlar.
Örnek olarak yukarıdaki resimde görüldüğü gibi 2 sayısını ifade eden tuş aynı zamanda A, B ve C harflerini de ifade eder normal şartlarda TextBox alanına sadece sayı girecek olursak 2 numaralı tuşa 4 defa basmamız gerekecektir. Bu 7 sayısı için 5 keredir. Bu durumda ekrana bir telefon numarası yazacağımızı varsayalım. Telefon numaramızda 07872787767 olsun bu durumda toplam 38 kere tuşlara basmamız gerekiyor. NUMERIC tipte bir constraints tipi tanımladığınız anda bu sayı 11 oluyor. NUMERIC tip her tuşun sadece sayı değeri taşıdığını belirtiyor bu durumda 2 tuşu A-B-C harfleri değil sadece 2 karaterini temsil ediyor.
Başka bir örnek olarak PHONENUMBER tipinde bir constraint sayı kontrolu yanında 12345678910 şeklinde girilen bir değeri 123 456 78910 şeklinde çevirir.

Screen nesnesinden türeyen tüm bileşenler gibi TextBox lardan da ekranda sadece bir adet bulunur. Ekrana atadığımız TextBox tüm ekranı kaplar ve diğer bileşenler için herhangi bir görüntüye izin vermez.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {

      TextBox ornek1= new TextBox("Ornek","",20,TextField.PHONENUMBER);
      TextBox ornek2= new TextBox("Ornek","",20,TextField.ANY);
      TextBox ornek3= new TextBox("Ornek","",20,TextField.NUMERIC);
      TextBox ornek4= new TextBox("Ornek","",20,TextField.DECIMAL);
      TextBox ornek5= new TextBox("Ornek","",20,TextField.EMAILADDR);
      TextBox ornek6= new TextBox("Ornek","",20,TextField.HYPERLINK);
      TextBox ornek7= new TextBox("Ornek","",20,TextField.PASSWORD);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(ornek1);
   }

   public void pauseApp() {
```

```
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki MIDlet örneğimizde 7 adet TextBox bulunuyor ve bunların hepsi ayrı Constraints değerlerine sahip ve bunlardan sadece ornek1 nesnesini ekranda gösteriyoruz.

TextBox Bilgilerine Ulaşmak

TextBox nesnesi üzerindeki değerlere ulaşmak için TextBox sınıfı içerisinde bazı metodlar mevcuttur. Örneğin başlık değiştirme, yazı değeri alma gibi işlemlerde get ve set metodları vasıtasıyla TextBox üzerinde istediğimiz değişikliği yapabilir veri ulaşımını sağlayabiliriz.

getString(), setString();

TextBox içerisindeki değerin değişimi veya erişimi için kullanılır. Aslında bu değer adındanda anlaşılabileceği gibi kurucu metod içerisinde girilen text değeridir. Nesneyi yarattıktan sonra da bu değeri değiştirme veya erişme imkanımız vardır.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
    public void startApp() {
        TextBox mesaj= new TextBox("Mesaj","Örnek Yazi",160,TextField.ANY);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(mesaj);
        System.out.println(mesaj.getString());
        mesaj.setString("Yeni Örnek Yazi");
        System.out.println(mesaj.getString());
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Örnek olarak startApp motodu içirisindeki yukarıdaki kod bloğunda mesaj adında bir TextBox yaratılmıştır. Text değeri olarak nesnenin yaratılması anında "Örnek Yazi" belirtilmiştir. Ancak *mesaj.setString("Yeni Örnek Yazi");* satırı ile bu değeri "Yeni Örnek Yazi" olarak değiştiriyoruz. İki ayrı kod bloğu olan *System.out.println(mesaj.getString());* satırları mesaj nesnesinin text içeriğini konsola yazdırır. Bu durumda birinci adımda konsol ekranına "Örnek Yazi" yazılacakken ikinci adımda "Yeni Örnek Yazi" yadırılır. Bu erişim sınıf içerisindeki herhangi bir yerden olabilir.

getSize(), setSize();

TextBox boyutları kurucu metodumuzdaki maxSize parametresi ile belirtilir bu parametreye kurucu metod haricinde *getSize()* ve *setSize(int size)* metodları ile de erişebiliriz.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {
      TextBox mesaj= new TextBox("Mesaj","Ornek Yazi",160,TextField.ANY);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(mesaj);
      System.out.println(mesaj.getMaxSize());
      mesaj.setMaxSize(100);
      System.out.println(mesaj.getMaxSize());
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Kod bloğumuzda içeriği “Örnek yazi” olan mesaj bir TextBox nesnemiz var. Öncelikle *System.out.println(mesaj.getMaxSize());* ile nesnemizin text boyutuna ulaşıyoruz. Başlangıç anında 160 verdiğimizden sonuç konsol ekranına 160 olarak bastırılıyor. *mesaj.setMaxSize(100);* satırında boyutu 100 olarak atıyoruz ve ikinci konsola yazdırma satırımızda değer 100 olarak dönüyor.

size

Bazen nesnemizin içerik boyutuna ihtiyaç duyabiliriz bu gibi durumlarda text metodu bize karakter uzunluğunu verir.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {
      TextBox mesaj= new TextBox("Mesaj","Ornek Yazi",160,TextField.ANY);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(mesaj);
      System.out.println(mesaj.size());
      mesaj.setString("Yeni Örnek Yazi");
      System.out.println(mesaj.size());
   }

   public void pauseApp() {
   }
```

```
    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki kodumuzda oluşturulan TextBox nesnesinin içerik uzunluğunu alıyoruz ilk adımda "Ornek Yazi" karakterinin uzunluğudur yani 10. *mesaj.setString("Yeni Örnek Yazi");* satırında TextBox nesnemizin içeriğini "Yeni Örnek Yazi" olarak değiştiriyoruz bu durumda ekrana size değerini bastırmak istediğimizde sonuç 15 olarak dönüyor. size metodu bir çok yerde kullanılabilir. Örnek olarak kullanıcıya veri girişi yaptırıken metin uzunluğunun istenen değerin üstünde veya altında olup olmadığını kontrol ettirebiliriz.

List

Kullanıcı girişlerinde sadece karakter girişleri yeterli olmayabilir. Bazı durumlarda kullanıcıya belirli değerleri seçtirmek isteyebiliriz işte böyle durumlarda List sınıfını kullanıyoruz. List bir dizi değer içerisinden bir ve bir kaçını seçmemize yarayan UI (User Interface) bileşenidir.
List yapısı aşağıdaki gibidir.

```
List (String title, int listType);
```

List sınıfının iki adet kurucu metodu vardır. Bunlar title ve listType tır.

title	List üzerinde görünecek başlık
listType	List in biçimi

Örnek bir list tanımlayacak olursak

```
List liste= new List("Ürünler",1);
```

Yukarıdaki örnekte liste adında ve List sınıfı tipinde bir nesne yarattık bu nesneye iki adet kurucu metod parametresi gönderdik "Ürünler" ve 1 bu parametrelere göre liste nesnemizin başlığı Ürünler tipide 1 olacaktır.
List içerisine veri eklemek için List.append() metodu kullanılır. Kullanım şekli aşağıdaki gibidir.

```
List.append(String stringPart, Image imagePart);
```

append() metodu sırasıyla iki adet parametre alır. Bunlar yazı (String) içeriği ve resim (Image) içeriği. Image yaratmak zorunlu değildir eğer elimizde bir resim yok ise Image nesnesi olarak null atayabiliriz. Bu durumda ekranda herhangi bir resim görünmeyecektir.
Kodumuzu örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
    public void startApp() {
```

```
        List liste= new List("Ürünler",List.EXCLUSIVE);
        liste.append("Bilgisayar", null);
        liste.append("Telefon", null);
        liste.append("DVD", null);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(liste);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Bu MIDlet in ekran görüntüsü aşağıdaki gibidir.

Göründüğü gibi üç adet ürün listelenmiş ve bunlardan birini seçme hakkı sağlanmıştır. Üst bölümde ise Ürünler başlığı bulunmaktadır. Image olarak null verdiğimiz için listemizde herhangi bir resim görünmüyor. Bunu resimli olarak göstermek istersek kodumuz aşağıdaki gibi olmalıdır.

```
import javax.microedition.midlet.*;
```

```
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {

      Image resim= null;
      try{
         resim=Image.createImage("/resim.jpg");
      }
      catch(Exception e){
         System.out.println("Resim oluşturulamadı");
      }
      List liste= new List("Ürünler",List.EXCLUSIVE);
      liste.append("Bilgisayar", resim);
      liste.append("Telefon", resim);
      liste.append("DVD", resim);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Resim yaratırken createImage() metodumuzu try-catch blokları arasına almamız gerekiyor aksi derleme anında uygulama derlense bile potansiyel hata durumunda olduğumuzu söyleyip derleme işlemini yapmayacaktır. Az öncekinden farklı olarak kodumuzda liste.append() derken imagePart alanına Image sınıfı tipindeki resim nesnesini gönderiyoruz.
Bu durumda uygulamanın ekran görüntüsü aşağıdaki gibi olur.

Göründüğü gibi listemizin yanında birer adet resim görünüyor. Bu resimler her satır için farklı birer Image olabilir ancak bu uygulamamızın bellek boyutunu artırabilir.

TextBox' ta olduğu gibi List sınıfında da bazı tip kolaylıkları sağlanmıştır. List sınıfının ikinci parametresi olan listType ile değişik tiplerde listeler yaratabiliriz.

IMPLICIT	Liste üzerinde gezinme
EXCLUSIVE	Tek seçim
MULTIPLE	Çoklu seçim

Bu değerler List sınıfının static değişkenleridir dolayısıyla değiştirilemez ve nesnelerini yaratmaya gerek yoktur. Herhangi bir yerde çağırmak için List.EXCLUSIVE şeklinde kullanırız.
Bu üç tipi tek tek deneyecek olursak

IMPLICIT

Bazen sadece liste şeklinde bir ekran ve sadece bir nesnenin seçilmesi ile işlemimizi gerçekleştirmek isteyebiliriz. Buna örnek telefon defterimiz olabilir. Daha detaylı örnek verecek olursak Nokia marka telefonlarda telefon defteri IMPLICIT tipinde bir List sınıfı nesnesidir tabi burada J2ME kullanılmamış olsa bile Java' nında bu kütüphaneleri kullandığını varsayarsak (Yüksek seviye bileşen olmasından dolayı) aslında aynı tip bileşenlerdir. Bu durumda bizde bu tarz bir telefon defteri için IMPLICIT tipinde List kullanırız.

```
import javax.microedition.midlet.*;
```

```
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {

      List liste= new List("Ürünler",List. IMPLICIT);
      liste.append("Bilgisayar", null);
      liste.append("Telefon", null);
      liste.append("DVD", null);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Yukarıdaki MIDlet içerisinde IMPLICIT tipinde bir List vardır. Bu MIDlet in ekran görüntüsü aşağıdaki gibi olur.

Görüldüğü gibi List sadece düz bir listeden oluşmakta ve seçtiğimiz anda işlem yapma olanağı sağlamaktadır. Bazı durumlarda form oluşturup adımlar ile ilermek veya birden fazla seçeneği işaretlemek isteyebiliriz böyle durumlarda bu tipi kullanamayız.

EXCLUSIVE

Adımlardan oluşan form doldurma gibi işlemlerde tek bir seçim yapmak istiyorsak EXCLUSIVE tipindeki List nesnelerini kullanırız. Bu nesneler web yada diğer uygulama ortamlarından tanıdığımız radiobutton tiplerine benzerler.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {

      List liste= new List("Ürünler",List.EXCLUSIVE);
      liste.append("Bilgisayar", null);
      liste.append("Telefon", null);
      liste.append("DVD", null);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Yukarıdaki MIDlet kodumuzu derleyip çalıştırırsak ekran görüntümüz aşağıdaki gibi olur.

Görüldüğü gibi sadece tek bir seçenek imkanı olan bir liste var. Bu liste ile bir adım seçilip sonraki adıma geçilebilir. IMPLICIT e göre farkı önce seçilmesi sonra işlemin yapılmasıdır.

MULTIPLE

Çoklu seçimler için MULTIPLE tipindeki List sınıflarını kullanırız. Bu bize adından da anlaşılacağı gibi birden çok seçim yapma imkanı sağlar. MULTIPLE nesnesi sıralı checkbox lar şeklindedir. Kullanım şekli diğer nesnelerle aynı olup sadece List tipini değiştirmek yeterlidir.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet {
   public void startApp() {

      List liste= new List("Ürünler",List.MULTIPLE);
      liste.append("Bilgisayar", null);
      liste.append("Telefon", null);
      liste.append("DVD", null);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);

   }
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki kodumuzun ekran çıktısı aşağıdaki gibidir.

Buradaki fark gördüğümüz gibi iki veya daha çok alanı işaretleyebilmemizdir. Aynı şekilde bu tip List nesnelerinde de resim kullanabiliriz.

Satır Yakalama

Seçilen satırların bulunması yada yakalanması işlemleri daha çok Command lar içerisinde kullanılır (Command ları ileriki adımlarda göreceğiz). Tek seçim ve çoklu seçimlerde farklı yakalama türleri vardır. Bunlar için List sınıfının bazı metodları kullanılır.

getSelectedIndex()

List içerisine eklediğimiz seçeneklerin hangilerinin işaretlendiği yada yakalandığını bulmak için List sınıfının getSelectedIndex() metodunu kullanırız. Bu metod hangi satırdaki elemanın seçildiğini döndürür. Satırlar 0 dan başlar ve satır sayısı boyunca devam eder. Ancak çoklu seçim imkanı sağlayan MULTIPLE tip liste sınıflarında sonuç her koşulda -1 yani herhangi bir seçim yapılmamıştır döndürebileceği değer tek bir satır olduğundan hangi satırı döndüreceğini bilemez. IMPLICIT ve EXCLUSIVE tiplerde böyle bir şey geçerli değildir.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet{
   List liste=null;
   public void startApp() {

      liste= new List("Ürünler",List.EXCLUSIVE);
      liste.append("Bilgisayar", null);
      liste.append("Telefon", null);
      liste.append("DVD", null);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);
      System.out.println(liste.getSelectedIndex());
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
   public void commandAction (Command c, Displayable s){
   }
}
```

Yukarıdaki kod bloğu bize List tipindeki liste nesnesinin hangi satırının seçili olduğunu gösterir. Ancak bu durumda 0 dönecektir çünkü standart olarak ilk eleman seçili gelir. Biz bu adımda herhangi bir seçim sonrası hareket (action) yapmadığımız için seçimimizde daha sonrada değişiklik yapsak dahi bu seçim yakalanamayacaktır. Bu konuyu Command' lar bölümünde işleyeceğiz.

getString(int elementNum)

Seçili satırların yakalanması sonrası içeriği alınmak istenebilir genel olarakta istenen budur. getString() metodu gönderdiğimiz satır numarasına (index) göre o satırın text içeriğini döndürür.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet{
   List liste=null;
   public void startApp() {

      liste= new List("Ürünler",List.EXCLUSIVE);
      liste.append("Bilgisayar", null);
      liste.append("Telefon", null);
      liste.append("DVD", null);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(liste);
```

```
        System.out.println(liste.getString(2));
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
    public void commandAction (Command c, Displayable s){
    }
}
```

Yukarıda *liste.getString(2)* ile 2 numaralı yani 3. sıradaki seçeneğin text değerini alıp konsola yazdırıyor.
Aslında seçim bu şekilde kod içine gömülerek değil seçili satır belirtilerek yapılır. Bu durumda

```
System.out.println(liste.getString(2));
```

yerine

```
System.out.println(liste.getString(liste.getSelectedIndex()));
```

yazmamız gerekiyor.

getImage(int elementNum)

İstenilen satırdaki resmi almak için getImage metodu kullanılır. Bu metod Image tipinde bir nesne döndürür ve bu seçili satırdaki imagePart' ın referansını gösterir.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class OrnekMIDlet extends MIDlet{
    List liste=null;
    public void startApp() {

        liste= new List("Ürünler",List.EXCLUSIVE);
        liste.append("Bilgisayar", null);
        liste.append("Telefon", null);
        liste.append("DVD", null);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(liste);
        Image resim= liste.getImage(0);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
```

```
    }
    public void commandAction (Command c, Displayable s){
    }
}
```

Image resim= liste.getImage(0) satırında ilk satırdaki resmi almış olduk. Aynı şekilde burada da *liste.getImage(liste.getSelectedIndex())* diyerek seçili satırı alabiliriz.

Alert

İşlem zamanlarında yada sonu gibi durumlarda kullanıcıya uyarı veya hata mesajları gibi bilgi ekranları gösterebilir yada sesli uyarı verebiliriz. Bu gibi durumlarda Alert nesnesi en çok işe yarayacak bileşenimizdir. Alert sadece görüntü olarak değil telefona ait ses özelliklerini kullanarakta kullanıcıya belli uyarılar yapabilir. Bunlar hata, uyarı, bilgi gibi durumlar olabilir.
Alert yapısı aşağıdaki gibidir.

```
Alert (String title);
Alert (String title, String messageString, Image alertImage,AlertType alertType);
```

Alert sınıfının iki adet kurucu metodu bulunur.
Birinci kurucu metod sadece String tipinde title parametresini alır. Bu parametre kullanıcıya sadece başlığı gösterir.
İkinci kurucu metod içerisinde ise 4 adet parametre bulunur. Birincisi ilkindeki gibi title yani başlık ikincisi String tipinde messageString yani Alert nesnemizin içiriğinde görünecek yazı; üçüncüsü eğer göstermek istiyorsak bir Image yani resim; dördüncü parametre de Alert nesnemizin tipidir. Herhangi bir resim göstermek istemiyorsak Image değeri olarak null göndeririz.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Uyari extends MIDlet {
    public void startApp() {
        Alert uyari= new Alert("Baslik Alani","Deneme Uyarisi",null,AlertType.ALARM);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(uyari);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örneğimizde Uyarı adında bir sınıfımız var. Bu sınıf içerisinde uyari adında bir Alert örneği tanımlıyoruz. Alert nesnemizin title yani başlığı “Baslik Alani”, içerisinde görünecek yazı “Deneme Uyarisi”, resmi null yani hiç bir resim görünmeyecek tipide ALARM. Bu durumda uygulamamızın akran çıktısı aşağıdaki gibi olur.

Görüldüğü gibi başlık alanımız ve içeriğimiz alert nesnesi üzerinde görünüyor. Alert özellikle işlem sonraları yada işlem anında çok işimize yarar.
AlertType bizim uyarılarımızın hangi tipte olacağını belirler.

ALARM	Alarm
CONFIRMATION	Doğrulama
ERROR	Hata
INFO	Bilgi
WARNING	Uyarı

Bu uyarıların görünümleri ve uyarı anında çıkan ses telefonlara göre değişebilir. Bileşenimiz yüksek seviye bir bileşen olduğundan telefonun kendi Alert özelliklerini kullanır.

setTimeout() getTimeout()

Uygulamamızda dikkat çekici bir nokta var. Çalışma anından kısa bir süre sonra alert nesnesi kayboluyor. Bunu default timeout süresi dışında bir değerin atanmamış olmasıdır. Normal şartlarda Alert belirli bir süre sonra ekrandan kaybolur. Bu default süreyi getDefaultTimeout() metodu ile alabiliriz. Örnek olarak emilatör üzerinde çıkan süre 2000 dir. Bu sayı milisaniye cinsinden olup 2 saniyeyi ifade etmektedir.
Timeout değerini **setTimeout** ile belirleyebiliriz.
setTimeout yapısı aşağıdaki gibidir.

```
setTimeout(int time);
getTimeout();
```

Bu örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Uyari extends MIDlet {
   public void startApp() {
      Alert uyari= new Alert("Uyari Yapiyorum","Deneme",null,AlertType.ALARM);
      System.out.println(uyari.getDefaultTimeout());
      System.out.println(uyari.getTimeout());
      uyari.setTimeout(10000);
      System.out.println(uyari.getTimeout());
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(uyari);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Yukarıdaki örnekte ilk adımda Alert nesnemizin default timeout değerini alıyoruz sonrasında ise belirli bir timeout değeri atayıp bunları her adımda konsol ekranına basıyoruz.

setString() getString()

Alert içerisindeki yazı değerlerine erişmek için kullanılır. Çalışma esnasında Alert içerikleri değiştirilmek istenebilir bu durumda setString ve getString metodları kullanılır.
Metodların yapısı aşağıdaki gibidir.

```
setString(String str);
getString();
```

Örnek vermek gerekirse

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Uyari extends MIDlet {
   public void startApp() {
      Alert uyari= new Alert("Uyari Yapiyorum","Deneme",null,AlertType.ALARM);
      System.out.println(uyari.getString());
      uyari.setString("Yeni Deneme");
      System.out.println(uyari.getString());
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(uyari);
   }
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Örnekteki ilk adımda Alert nesnesinin içerik değeri “Deneme” olacaktır. Sonraki adımlarda değer “Yeni Deneme” yapılır ve her adımda içerik konsol ekranına bastırılacaktır.

setType() getType()

Alert nesnemizin tipini de nesnemizi oluşturduktan sonra değiştirebiliriz. Bunu setType ve getType metodları sağlar.
Metodlarımızın yapısı aşağıdaki gibidir.

```
setType(AlertType type);
getType();
```

setType içerisine AlertType sınıfının içerisinde bulunan daha önce bahsettiğimiz ALARM gibi tipler gönderilir ve getType ile bu tipler alınabilir.

Örnek olarak

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Uyari extends MIDlet {
    public void startApp() {
        Alert uyari= new Alert("Uyari Yapiyorum","Deneme",null,AlertType.ALARM);
        uyari.setType(AlertType.ERROR);
        uyari.getType();
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(uyari);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki adımda ALARM tipindeki bir Alert ERROR tipine dönüştürülmüştür. Gerçek hayattan örnek verecek olursak uygulamamızın nasıl sonlandığına göre uyarı tipimize değiştirebiliriz. Örnek olarak uygulama başarılı olarak tamamlanmışsa INFO hata vermişse ERROR tipinde olabilir.

getTitle setTitle

Diğer setTitle ve getTitle' lar ile aynı işleve sahiptir Alert nesnesinin başlık alanına erişmek için kullanılır.
Metodun Yapısı

```
setTitle(String s);
getTitle();
```

Kodu örneklendirmek gerekirse

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Uyari extends MIDlet {
    public void startApp() {
        Alert uyari= new Alert("Uyari Yapiyorum","Deneme",null,AlertType.ALARM);
        System.out.println(uyari.getTitle());
        uyari.setTitle("Baslik Degisti");
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(uyari);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki kodda uygulamamızın başlığı nesne yaratılırken belirleniyor sonrasında ise değer değiştiriliyor. Değer değiştirilmeden önce konsol ekranına getTitle() metodu ile basılıyor.

Form

Kullanıcı arayüzü (UI – User Interface) bileşenlerinde şimdiye kadar gördüğümüz kadarı ile sadece bir adet nesneyi ekrana atayabiliyorduk. Peki şöyle bir senaryomuz olsun bizden mobil tabanlı bir öğrenci bilgi sistemi yapmamız isteniyor ve kullanıcıdan 16 adet parametre alınacak. Bu gibi bir durumda Command' lar kullanılarak isteğe göre adımlarda ekrana diğer bileşen aktarılabilir fakat bu hem kullanıcı için geçişi zor hemde göz önünde olmayan bir yapı çıkarır. İşte böyle birden çok veri girişinin olduğu durumlarda **Form** sınıfı kullanılır.

Formlarda kullanıcı yarattığı her Form bileşenini önce forma ekler sonrasında form nesnesi ekran nesnesine aktarılır. Bu durumda yine ekrana tek bir bileşen atanmış olur.
Form nesnesinin yapısı aşağıdaki gibir.

```
Form(String title);
Form(String title, Item[] items);
```

Form sınıfı içerisinde String tipinde title parametresi ile yine title ile itmes adında form bileşenlerini alan iki adet kurucu metod vardır. Title ile formun başlığını belirleyebiliyoruz, items ilede elimizde bulunan bileşenleri forma tek seferde ekleyebiliyoruz.

Form için bir örnek yapalım.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Bu kodun ekran çıktısı aşağıdaki gibi olur.

Göründüğü gibi ekranda sadece başlığımız görünüyor bunun dışında sadece boş beyaz bir alan mevcut. İşte bu durumda forma yeni form bileşenleri atamak gerekiyor. Form üzerine sadece forma özgü bileşenler atanabilir TextBox, List gibi bir bileşenler yerine alternatif form bileşenleri mevcuttur.

TextField

TextBox benzeri bir form bileşenidir tek başına ekrana atanamaz sadece form bileşenine eklenebilir. Form bileşeninin ekrana atanmasıyla ekranda görünür. Bir form üzerinde birden fazla TextField bileşeni olabilir.
Yapı olarak aşağıdaki gibidir.

```
TextField(String label, String text, int maxSize, int constraints);
```

Label değişkeni TextField nesnesinin yanında yada üstünde bulunan etikettir. Bu bir form için "Ad" veya "Soyad" gibi bir form etiketi olabilir. text parametresi TextField içerisinde görünecek değerdir. maxSize TextField' ın alabileceği maksimum değer, constraints ise textfield tipidir. Göründüğü gibi bu değerler TextBox ile hemen hemen aynıdır. Sadece ilk parametre TextBox için title' dır.
TextField sınıfı için bir örnek yapalım.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        TextField ad= new TextField("Adiniz:","",20,TextField.ANY);
        kayitFormu.append(ad);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki kodun ekran çıktısı aşağıdaki gibidir.

Göründüğü gibi tek bir satırda bulunan bir TextField nesnesi oluştu. Eğer nesnemiz TextBox olsaydı tüm ekranı kaplayacaktı.
Kodumuzda *kayitFormu.append(ad);* satırı olmasaydı ekranda TextField nesnesi görünmeyecekti. Nesne oluşmuş fakat forma eklenmemiş olacaktı.
Bir Form üzerinde birden fazla TextField olabilir demiştik bunu örneklendirelim.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
   public void startApp() {
      Form kayitFormu= new Form("Kayit Formu");
      TextField ad= new TextField("Adiniz:","",20,TextField.ANY);
      TextField soyad= new TextField("Soyadiniz:","",20,TextField.ANY);
      TextField yas= new TextField("Yasiniz:","",20,TextField.NUMERIC);
      kayitFormu.append(ad);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(kayitFormu);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
```

```
}
```

Formumuza ad, soyad ve yas adında üç adet TextField bileşeni ekledik. Kodumuzun çıktısı aşağıdaki gibi olacaktır.

Göründüğü gibi ekranımızda üç adet TextField bileşeni göründü bunlardan yas TextFiled.NUMERIC constraints özelliğine sahip bu yüzden sadece sayı tipinde veri girişi yapılabilir. Bu özellikler TextBox contraints özellikleri ile aynıdır.

ANY	Herhangi bir karakter
EMAILADDR	E-mail adresi
NUMERIC	Sayı
PHONENUMBER	Telefon numarası
URL	İnternet adresi
DECIMAL	Ondalıklı
PASSWORD	Şifre

Bu özellikler TextField nesnesinin static ve final değişkenleridir bu yüzden nesnelerinin yaratılmasına gerek yoktur ve değerleri değiştirilemez.

setLabel() getLabel()

Label yani etiketlere erişmek için kullanılan metodlardır. Çalışma anında etiketler değiştirilebilir yada değerleri alınabilir.
Metodların yapısı aşağıdaki gibidir.

```
setLabel(String label);
getLabel();
```

setLabel(); metodu label adında String tipinde bir değişken alır. getLabel() ise label içeriğine erişmek için kullanılır.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
   public void startApp() {
      Form kayitFormu= new Form("Kayit Formu");
      TextField ad= new TextField("Adiniz:","",20,TextField.ANY);
      System.out.println(ad.getLabel());
      ad.setLabel("Ad:");
      System.out.println(ad.getLabel());
      kayitFormu.append(ad);
      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(kayitFormu);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Yukarıdaki örnekte ad isimli TextField nesnesinin label değeri nesne yaratılırken “Adiniz:” olarak veriliyor ancak daha sonra “Ad:” olarak değiştiriliyor. İşlem aralarında label değerleri konsol ekranına bastırılıyor.

setString() getString()

TextField içerisindeki değerlerin çalışma anında değiştirilmesini sağlar. Örnek olarak kontrol yaptığımız bir alanı setString() ile temizleyebiliriz.
Metodların yapısı aşağıdaki gibidir.

```
setString(String text);
getString();
```

setString() veriyi değiştirmek getString() ise var olan veriyi almak için kullanılır. Örnek olarak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
   public void startApp() {
      Form kayitFormu= new Form("Kayit Formu");
```

```
        TextField ad= new TextField("Adiniz:","",20,TextField.ANY);
        System.out.println(ad.getString());
        ad.setString("Melih");
        System.out.println(ad.getString());
        kayitFormu.append(ad);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukardaki kod bloğunda TextField nesnesini oluşturduğumuz anda içerisindeki değer boş olarak veriliyor. Sonraki adımda içerideki değer “Melih” olarak değiştiriliyor.

DateField

Tarih girişleri yazılımcılar için her zaman sorun olmuştur. Tarih girişi için kullanıcı dd/mm/yyyy, dd.mm.yyyy, dd/mm/yy gibi formatlar kullanabilir. TextField gibi alanlarda bunu anlamamız sorun olur sonuç olarak girilen ilk veya ilk iki değer gün yada ay olabilir. DateField J2ME ortamında bu girişleri konutrollü bir şekilde yapmamızı sağlar. Bu telefonun özelliklerine göre formatlı bir giriş ekranı yada takvim nesnesi olabilir .
Kullanım şekli:

```
public DateField(String label, int mode);
```

Label, TextField’ ta olduğu gibi input alanının açıklama etiketini oluşturur. Mode giriş ekranımızın hangi tipte olacağıdır.
Bileşenimizi kod olarak örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        DateField tarih= new DateField("Tarih",DateField.DATE);
        kayitFormu.append(tarih);
        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
```

```
    }
}
```

Uygulamamızın çıktısı aşağıdaki gibi olur.

DateField bileşeni form ekranımızda yukarıdaki gibi görünür ancak giriş yapılmak istendiğinde aşağıdaki şekli alır. Emilatör ortamında böyle görünmesine karşın yüksek seviye bir bileşen olduğundan farklı telefonlarda farklı şekillerde görünebilir. Örnek olarak bu çıktı ericsson ortamında çok daha farklıdır.

Mod tiplerini yine DateField sınıfından alabiliriz.

DateField.TIME	Saat
DateField.DATE	Tarih
DateField.DATE_TIME	Tarih ve Saat

Yukarıdaki mod seçenekleri ile veri girişlerimizi daha kullanılabilir hale getirebiliriz. Veri girişleri sonrasında Date tipinde bir değişken döner.
DateField mod seçeneklerini tek tek inceleyecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
   public void startApp() {
      Form kayitFormu= new Form("Kayit Formu");
      DateField tarih1= new DateField("Tarih",DateField.TIME);
      DateField tarih2= new DateField("Tarih",DateField.DATE_TIME);
      DateField tarih3= new DateField("Tarih",DateField.DATE);
      kayitFormu.append(tarih);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(kayitFormu);
   }
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

DateField.TIME

DateField.DATE

DateField.DATE_TIME

DATE_TIME mod her iki veri girişinede imkan sağladığından seçilen Date ve Time liklerinde iki tip bileşende görünecektir.

setLabel() getLabel()
Nesnenin label özelliğine erişmeyi sağlarlar.

```
setLabel(String label);
getLabel();
```

setInputMode() getInputMode()
DateField nesnesinin mode özelliğine erişmeye yarar. Mode TIME, DATE veya DATE_TIME olabilir.

```
setInputMode(int mode);
getInputMode();
```

setInputMode() metodu içerisinde atanan int değer DateField sınıfından alınabilir. Örnek verecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
   public void startApp() {
      Form kayitFormu= new Form("Kayit Formu");
      DateField tarih= new DateField("Tarih",DateField.TIME);
      tarih.getInputMode();
      tarih.setInputMode(DateField.DATE);

      kayitFormu.append(tarih);

      Display ekran=Display.getDisplay(this);
      ekran.setCurrent(kayitFormu);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

ChoiceGroup

Çoklu seçimlerde daha önceden List sınıfını kullandığımızı görmüştük. Bu sınıfta List nesnesinin hemen hemen tüm özelliklerini taşımaktadır. Zaten iki sınıfta Choice arayüzünden türemişlerdir. Bu bileşen aynen List sınıfı gibi veri girişi yerine seçim imkanı sağlar. Bu seçim bir veya birden çok olabilir.

ChoiceGrup yapısı aşağıdaki gibidir.

```
ChoiceGroup(String label, int choiceType);
ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements);
```

Sınıf yapısında görüldüğü gibi değişkenler List sınıfı ile aynı sadece ilk değişken title yerine label olarak geliyor. Bu sayede form üzerinde bileşenin yanında açıklamasını ekleme imkanına sahip oluyoruz.

Bileşenimizi kod olarak örneklemek isteyecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.EXCLUSIVE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);

        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örnekte bir sehir adında bir nesne oluşturduk ve tipini ChoiceGroup.EXCLUSIVE yaptık. Nesnemize üç adet değer atadık ve kullanıcımızdan bunlardan birini seçmesini bekliyoruz. Örnek kodumuzun çıktısı aşağıdaki gibidir.

ChoiceGroup tipleri List sınıfındaki gibidir. Ancak buda IMPILICT kullanilmaz.

EXCLUSIVE Tek seçim
MULTIPLE Çoklu seçim

MULTIPLE
Çoklu seçimler için kullanılır. Örnek olarak ders seçimi yapınız. Burada birden çok ders seçilmek istenebilir.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.MULTIPLE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);

        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }
```

```
    public void destroyApp(boolean unconditional) {
    }
}
```

Uygulamanın ekran çıktısı aşağıdaki gibidir.

EXCLUSIVE
Tek bir seçim için kullanılır programlama dünyasında radio buton olarakta adlandırılır. Örnek olarak şu anda bulunduğunuz şehiri seçiniz gibi.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.EXCLUSIVE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);

        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Uygulamımızın ekran çıktısı aşağıdaki gibidir.

getSelectedIndex() setSelectedIndex()
Seçilen satırı bulma yada bir satır seçmek için kullanılır. Örnek olarak o an için seçili satırı alabilir yada bir satırı seçilmiş olarak işaretlemek isteyebiliriz.
Metod yapısı aşağıdaki gibidir.

```
getSelectedIndex();
setSelectedIndex(int elementNum, boolean selected)
```

getSelected seçili olan satırın değerini int tipinde döner. setSelected ise iki adet parametre alır. Bunlar elementNum yani seçilecek olan satırın sırası ve seçili olup olmayacağıdır yani bu komut ile seçili olan bir satırın işaretinide kaldırabiliriz. elementNum parametresinde gönderilecek satırlar 0 dan başlar.
Metodlarımızı örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
```

```
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.EXCLUSIVE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);
        System.out.println(sehir.getSelectedIndex());
        sehir.setSelectedIndex(1,true);
        System.out.println(sehir.getSelectedIndex());
        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Örnek kod bloğumuzda *System.out.println(sehir.getSelectedIndex())* ile seçili olan satır alınıyor burada herhangi bir seçim işlemi yapılmadığından standart olarak ilk satır seçili geliyor bu durumda dönüş değeri 0' dır. *sehir.setSelectedIndex(1,true)* ile 1 numaralı indexe sahip satır seçili hale getiriliyor bu baştan ikinci satır anlamına gelmektedir. Tekrardan *System.out.println(sehir.getSelectedIndex())* ile seçili satırı getirmek istediğimizde bize 1 değerini verecektir. Yani daha öncede set metodu ile belirlediğimiz satır.

getString()

Seçili olan satırın içeriğini verecektir. Bu sayede kullanıcıdan index yerine seçili satırın text içeriğini alabiliriz.
Metod yapısı aşağıdaki gibidir.

```
sehir.getString(int elementNum)
```

Parametre olarak değerin alınmak istediği satırın index değerini istemektedir. Yani 1 değeri verirsek baştan ikinci satırın text değerini vermiş olur.
Örnek verecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.EXCLUSIVE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);

        System.out.println(sehir.getString(1));
        sehir.setSelectedIndex(2, true);
        System.out.println(sehir.getString(sehir.getSelectedIndex()));
```

```
        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örneğimizde *System.out.println(sehir.getString(1))* ile 1 numarı indexteki satırın text değerini konsol ekranına basıyoruz. *sehir.setSelectedIndex(2, true)* satırı ile 2 numaralı index satırını seçili hale getirdik. Bu durumda ekranda baştan 3. satır seçili olacaktır. *System.out.println(sehir.getString(sehir.getSelectedIndex()));* ile seçili olan yani baştan 3. satırın text değerini konsol ekranına basıyoruz.

setLabel() getLabel()

Etiket değerlerine erişmek için kullanılır. Çalışma anında bir ChoiceGroup nesnesinin etiketini alabilir yada değiştirebiliriz.
Metod yapısı aşağıdaki gibidir.

```
setLabel(String label)
getLabel()
```

Örnek verecek olursak

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        ChoiceGroup      sehir=new      ChoiceGroup("Sehir      Seciniz",
ChoiceGroup.EXCLUSIVE);
        sehir.append("Ankara",null);
        sehir.append("Istanbul",null);
        sehir.append("Izmir",null);

        System.out.println(sehir.getLabel());
        sehir.setLabel("Sehirler");
        System.out.println(sehir.getLabel());

        kayitFormu.append(sehir);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
```

```
    }
}
```

Yukarıdaki örnekte System.*out.println(sehir.getLabel())* ile sehir nesnesinin etiket değerini konsol ekranına yazdırdırk daha sonra *sehir.setLabel("Sehirler")* ile etiket değerini “Sehirler” yapıyoruz ve tekrardan değerimi konsol ekranına bastırıyoruz.

Gauge

Ölçü benzeri girişler için kullanılır. Veri girişi esnasında gauge nesnesi belli limitler arasında hareket ettirilir. Sonuc almak istediğimiz zaman belirlediğimiz limitler dahilinde bir sonuç döner. Görünün aygıtlar arasında farklılık gösterebilir.
Sınıf yapısı aşağıdaki gibidir.

```
Gauge(String label, boolean interactive, int maxValue, int initialValue)
```

Label bileşenin açıklamasını oluşturacak etikettir. interactive boolean tipinde bir değişkendir yani iki tip değer alabilir true veya false. Değer true olursa bileşen üzerinde değer değişikliği yapabiliriz, false durumunda değişikliğe izin verilmez. Üst değer atamak için maxValue kullanılır örnek olarak bu değeri 20 atarsak gauge üzerinde 20 hareketlik bir değer aralığı oluşur. İlk değeri belirlemek içinse initialValue kullanılır 20 üst limiti verilen bir gauge nesnesinde initialValue 4 verilirse sadece ilk blok seçili olacaktır.
Bileşenimizi örnekleyelim.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        Gauge deger= new Gauge("Deger",true,5,2);
        kayitFormu.append(deger);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örnekte deger isminde Gauge tipinde bir nesne oluşturuluyor sonrasında nesne kayitFormu isimli forma ekleniyor. Gauge bileşeni 4 adet değer alıyor. Bileşenin etiketi “Deger”, aktiflik durumu true yani olumlu ve aktif, üst limiti 5 ve ilk değeri 2. Bu durumda uygulamamızın ekran çıktısı aşağıdaki gibidir.

Ekran çıktısında görüldüğü gibi nesnede sadece ilk iki blok seçili bunun sebebi üst limitin 5, ilk değerin 2 verilmesidir.

getValue() setValue()

Gauge nesnesi üzerindeki değerlere erişim için kullanılır. Çalışma anında bir değere değiştirmek yada almak isteyebiliriz bu durumda value değişkeninin get ve set metodları kullanılır.

Metod yapısı

```
getValue();
setValue(int value);
```

Metodlarımızı örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        Gauge deger= new Gauge("Deger",true,5,2);
        kayitFormu.append(deger);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
        System.out.println(deger.getValue());
        deger.setValue(4);
        System.out.println(deger.getValue());
```

```
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
}
```

Örnekte ilk değer olarak 2 atanıyor ve form ekranda gösterildikten sonra Gauge değeri konsol ekranına yazdırılıyor. Bu durumda değer 2 olarak yazdırılacaktır. Sonrasında `deger.setValue(4)` ile Gauge değeri 4 olarak veriliyor ve getValue() ile alınıp konsola yazdırılıyor.

getMaxValue() setMaxValue()
Üst limit değerlerine erişim için kullanılır. MaxValue değerine get ve set metodları ile ulaşılabilir.
Metod yapısı

```
getMaxValue();
setMaxValue(int maxValue);
```

Metodumuzu örneklendirecek olursak.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormOrnek extends MIDlet {
    public void startApp() {
        Form kayitFormu= new Form("Kayit Formu");
        Gauge deger= new Gauge("Deger",true,5,2);
        kayitFormu.append(deger);

        Display ekran=Display.getDisplay(this);
        ekran.setCurrent(kayitFormu);
        System.out.println(deger.getMaxValue());
        deger.setMaxValue(20);
        System.out.println(deger.getMaxValue());
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örnekte ekrana aktarılan Gauge nesnesinin ilk üst limiti ilk olarak 5 verilmiştir ve *deger.getMaxValue()* ile çağırıldığında

Ticker

Ticker direk olarak ekrana eklenen bir UI (User Interface) bileşeni değildir. Bu bileşenlere ek olarak kullanılan bir özelliktir. Bu sınıf ile TextBox, List, Form gibi bileşenlere ek olarak kayan yazı tarzında bir özellik ekleyebiliriz. Bir arayüz objesi üzerinden örnek verecek olursak setTicker yöntemiyle objeye eklenir. setTicker içerisinde yeni bir ticker nesnesi yaratmalıyız.
Kod bloğu aşağıdaki gibidir.

```
setTicker(Ticker ticker);
```

Bunu TextBox nesnesi üzerinde örneklendirecek olursak aşağıdaki gibi olur.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TickerOrnek extends MIDlet {
    public void startApp() {
        TextBox    mesaj    =    new    TextBox("Mesajiniz","Burasi    TextBox
Icerigi",100,TextField.ANY);
        Ticker kayanYazi = new Ticker("Mehaba Dünya");
        mesaj.setTicker(kayanYazi);
        Display.getDisplay(this).setCurrent(mesaj);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Yukarıdaki örnekte mesaj adına yeni bir TextBox nesnesi yaratılıyor ve bu TextBox' a kayan yazı adında bir Ticker nesnesi set ediliyor. Bu tüm UI nesneleri için geçerlidir. Ayrıca örneğimizdeki gibi nesneyi bir üst ayrıca yaratmak yerine *mesaj.setTicker(new Ticker("Merhaba Dünya"))* şeklinde de ekleyelibiliriz.