Zemberek Projesi Sözlük Yapısı ve Kök Seçicileri

Zemberek projesinde Sözlük ile kastedilen şey “**Kök Sözlüğü**” dür. Bu dokümanın hazırlandığı sırada zemberek kütüphanesinin kullandığı kök sözlüğü yaklaşık 30 bin kök içeriyordu.

Sözlük Okuyucu

Çeşitli tiplerdeki sözlük dosyalarını okuyan sınıflar bu arayüzü gerçeklemelidirler. Şu anda iki adet gerçekleyici sınıfı vardır, **BinarySozlukOkuyucu** ve **DuzMetinSozlukOkuyucu**. Sözlük Okuyucular `net.zemberek.bilgi.araclar` paketinde bulunur

BinarySozlukOkuyucu sınıfı ikili (Binary) olarak düzenlenmiş sözlüğü okur. İkili sözlüğe özellikle Zemberek Open Office eklentisinin daha hızlı açılabilmesi için gerek duyulmuştur. Okunan veriler işlenmeden doğrudan tablolara bakılarak kökler oluşturulduğu için düzyazı sözlüğü okumaktan daha hızlıdır. Constructor'lardan birincisi sözlük dosyasının adını kullanırken ikincisi de Open Office gibi URL'lerle çalışan uygulamalarda kullanılabilir.

```
public interface SozlukOkuyucu
{
    public void initialize(String fileName);

    public void initialize(URI fileURL);

    public List hepsiniOku();

    public Kok oku();
}
```

Burada kullanılan ana metod **oku()** dur. Açılan sözlük dosyasında okunacak kök kalmayana dek bu metodu çağırarak tüm sözlüğü okuyabilirsiniz. Metod, okunacak kök kalmayınca null döndürür. Eğer tüm kökleri liste olarak okumak isterseniz **hepsiniOku()** metodu kullanılabilir.

Kod Örneği:

```
...
SozlukOkuyucu okuyucu = new BinarySozlukOkuyucu();
okuyucu.initialize("kaynaklar/kb/binary-sozluk.bin");
Kok kok = null;
while((kok = sozlukOkuyucu.oku())!= null){
     ekle(kok); // Elde edilen kök nesnesi ile ne gerekiyorsa yap.
}
...
```

Sözlük Yazıcı

Sözlük okuyucu gibi sözlük yazıcılar da benzeri bir arayüzü gerçeklemek zorundadırlar. İkili(Binary) ve düzyazı sözlükler için iki adet SözlükYazıcı bulunur.

```
public interface SozlukYazici
{
    public void initialize(String fileName);

    public void initialize(String fileName, String encoding);

    public void yaz(List kelimeler);

    public void ekle(List kelimeler);

    public void kapat();
```

```
}
```

Kod Örneği : Aşağıdaki kod Bir düzyazı sözlüğü okur ve Binary sözlüğe dönüştürür.

```
SozlukOkuyucu okuyucu = new DuzMetinSozlukOkuyucu();
SozlukYazici yazici = new BinarySozlukYazici();
okuyucu.initialize("duzyazisozluk.txt");
yazici.initialize("binarysozluk.bin");
List list = okuyucu.hepsiniOku();
yazici.ekle(list);
yazici.kapat();
```

Sözlük Yapısı

Zemberek kök sözlükleri **/kaynaklar/kbsource/duzyazi-kilavuz.txt** ve 6000 civarında insan ismi taşıyan **kaynaklar/kbsource/ozelisimler/kisi-adlari.txt** dosyaları baz alınarak oluşturulur. Kök sözlüğünü oluşturmak için "net.zemberek.bilgi.araclar.SozlukAraclari" sınıfı kullanılır. Bu sınıf kendi başına çalıştırıldığı zaman zembereğin kullandığı binary ve düzmetin sözlükleri oluşturacaktır. Bu sözlükler "**kaynaklar/kb/duzyazi-sozluk.txt ve kaynaklar/kb/binary-sozluk.bin"** dosyalarıdır. duzyazi-kilavuz dosyası ile duzyazi-sozluk.txt dosyalarının kullandıkları formatlar aynıdır.

Bu dosyada kullanılan özel karakter ve kısaltmaların listesi:

Kısaltma	***Anlam***
#	*Yorum. Bu satırlar okuyucular ihmal edilir.*
IS	*İsim*
FI	*Fiil*
SI	*Sıfat*
SA	*Sayı*
OZ	*Özel isim*
ZA	*Zamir*
YUM	*Yumuşama. Örnek: kitap – kitabı (kitapı değil)*
DUS	*Harf Düşmesi nutuk – nutka (nutuka değil)*
TERS	*Ters dönüşüm saat – saate (saata değil)*
YAL	*Kelime sadece yalın olarak kullanılır*
GEN	*Geniş zaman istisnası*
KESMESIZ	
KAYNASTIRMA_N	
OZEL_IC_KARAKTER	
IS_TAM	
EK	

Tartışma: Sözlükteki her kelimeyi özel olarak işaretlemek yerine bazı istisnai durumlar için bu istisnaların kurallara göre otomatik olarak işaretlenmesi mümkün olabilir.

Sözlükten bir parça:

```
#Sözlük parçası
```

```
zeki SI
zelzele IS
zem IS CIFT
zemberek IS YUM
zembil IS
zemheri IS
zemin IS
zemmet FI GEN YUM
```

Sözlükler (Sözlük.java)

Kök Sözlüğünün sınıf-nesne olarak ifadesi olan tüm sözlük sınıfları Sözlük arayüzünden türetilmelidirler. Sözlük Arayüzü:

```
public interface Sozluk
{
    /**
     * Verilen kelime için seklinde yazilan tum kelime koklerini dondurur.
     * giriş, kokun istisna halini taşıyan bir kelime de olabilir.
     * @return kok listesi.
     */
    public Collection kokBul(String str);

    /**
     * sozluk icindeki normal ya da kok ozel durumu seklindeki
     * tüm kok iceriklerini bir Koleksiyon nesnesi olarak dondurur.
     * @return : Tüm kökleri taşıyan liste
     */
    public Collection getKelimeler();

    /**
     * sozluge kok ekler.
     * @param kok
     */
    public void ekle(Kok kok);

    public KokSeciciFactory getKokSeciciFactory();
}
```

Bu arayüzde ekle(Kok) ve getKokSeciciFactory() dir. Genel olarak Sözlük arayüzünü gerçekleyen sınıflar

- Bir sözlük okuyucu oluştur
- Kökleri oku
- Veriyapısına yerleştir

adımlarını gerçeklerler. Şu anda Zemberek kütüphanesinde yalnızca bir Sözlük gerçeklemesi vardır, Ağaç Sözlük (TreeSozluk.java)

getKokSeciciFactory() ise Çözümleyiciler tarafından kök adaylarının belirlenmesi için kullanılan KökSeçiciler için bir fabrika gerçeklemesi döndürürler. Bu yapının detayları KökSeçiciler bölümünde anlatılacaktır.

AğaçSözlük (TreeSozluk.java)

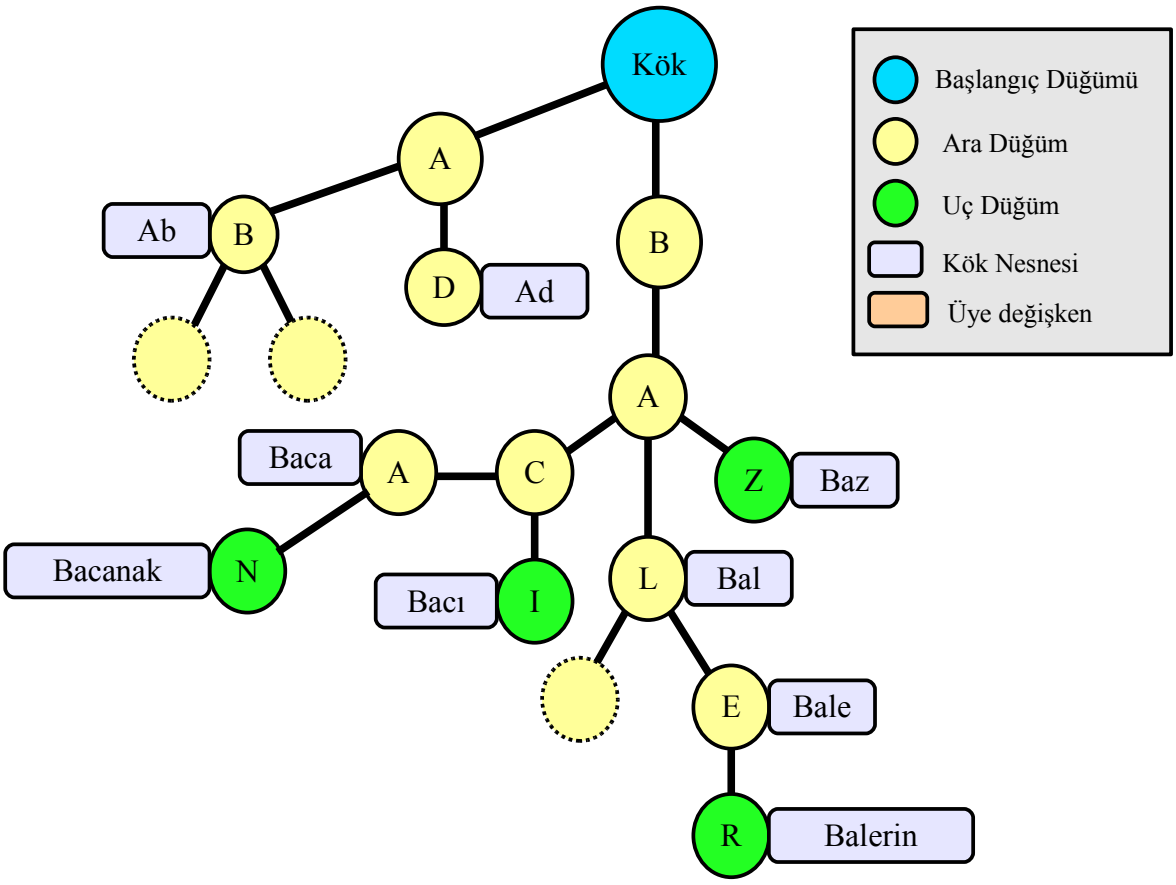
Zemberek kütüphanesindeki tek sözlük gerçeklemesi Ağaç Sözlüktür. Daha önce yapılan iki gerçeklemeler istenirse cvs sunucusundaki attic'den elde edilebilir. Ağaç Sözlük kendisine verilen bir sözlük okuyucuyu kullanarak tüm kökleri okur ve bir Kök Ağacına yerleştirir. Belli bir kelime için aday kökleri bulmak isteyen istemciler Ağaç sözlükten bir KökSeçici Üretici (KokSeciciFactory) isterler ve bu factory yardımı ile istedikleri türdeki

kökseçiciyi elde ederler. Yani Sözlük aynı zamanda KökSeçicileri oluşturmak için kullanılan fabrikaya da erişim noktasıdır.

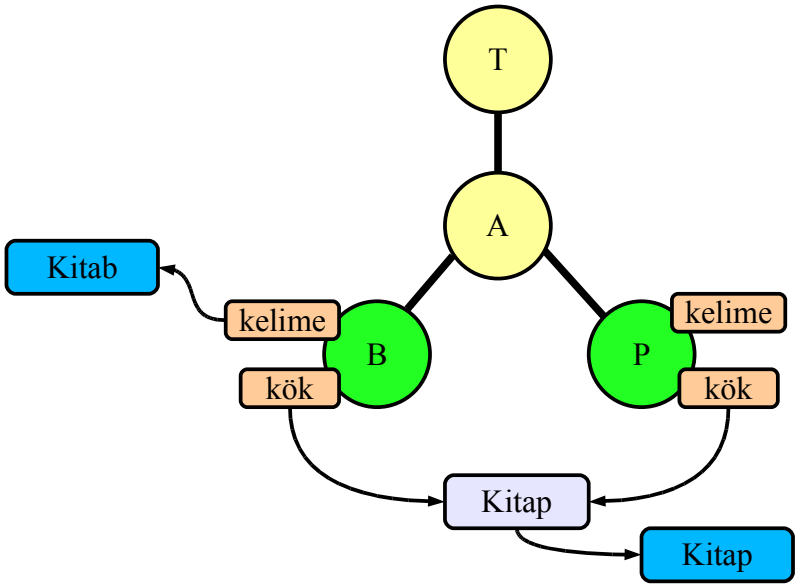
Kök Ağacı (KokAgaci.java)

Kök ağacı zemberek sisteminin temel veri taşıyıcılarından biridir. Kök sözlüğünden okunan tüm kökler bu ağaca yerleştirilirler. Ağacın oluşumundan **TreeSozluk** sınıfı sorumludur. Kök ağacı kompakt DAWG (Directed Acyclic Word Graph) – Patricia tree benzeri bir yapıya sahiptir. Ağaca eklenen her kök harflerine göre bir ağaç oluşturacak şekilde yerleştirilir. Bir kök'ü bulmak için ağacın başından itibaren kök'ü oluşturan harfleri temsil eden düğümleri izlemek yeterlidir. Örneğin aşağıdaki şekilde “Bale” kökünü bulmak için önce ağacın en başındaki “b” harfine, sonra “a” , sonra “l” ve son olarak ta “e” harfini temsil eden alt düğüme doğru ilerlemek yeterlidir. Eğer bir kök'ü ararken erişmek istediğimiz harfe ait bir alt düğüme gidemiyorsak kök ağaçta yok demektir.

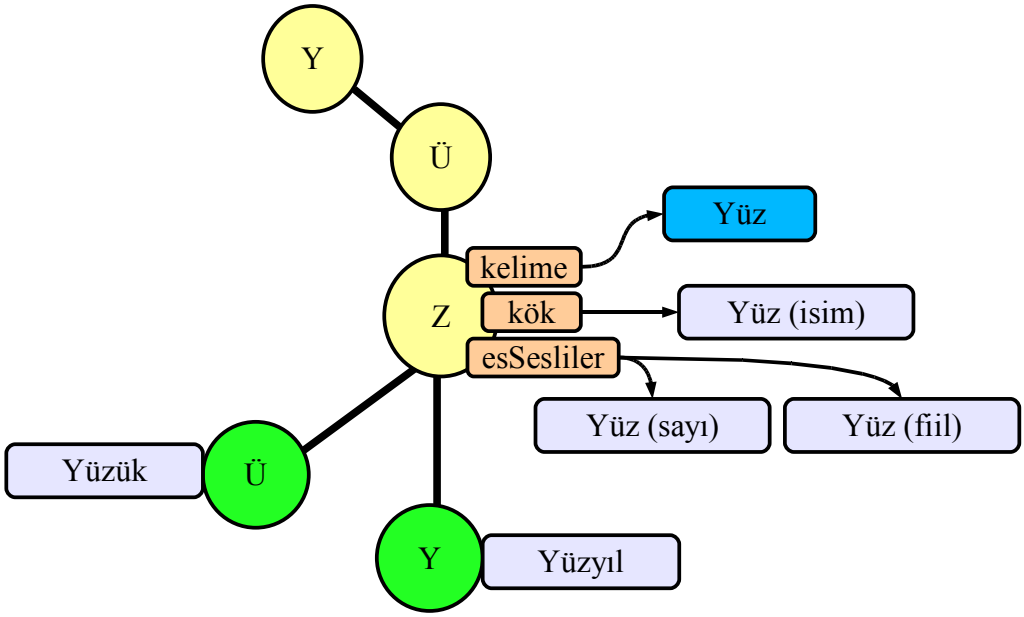
Ağacın bir özelliği de boşuna düğüm oluşturmamasıdır. Eğer bir kök'ün altında başka bir kök olmayacaksa tüm harfleri için ayrı ayrı değil, sadece gerektiği kadar düğüm oluşturulur. Aşağıdaki şekilde “balerin” kökü için sadece B-A-L-E-R düğümleri oluşmuş, daha aşağıda kök olmadığı için Balerin kökü doğrudan “R” düğümüne eklenmiştir.



Çeşitli nedenlerle değişikliğe uğrayabilecek olan kökler ağaca eklenirken değişmiş halleri ile beraber eklenirler. Örneğin **kitap** kökü hem “**kitab**” hem de “**kitap**” hali ile sözlüğe eklenir, ancak bu iki kelime için oluşan düğüm de aynı kök'ü gösterirler. Böylece “Kitabına” gibi kelimeler için kök adayları aranırken “kitap” köküne erişilmiş olur.



Eş Sesli olan kökler aynı düğüme bağlanırlar. Aşağıdaki örnekte "**Yüz**" kökünün üç hali de aynı köke bağlanmış şekilde duruyorlar. Ağacın oluşumu sırasında ilk gelen kök düğümdeki "**kök**" değerine, sonradan gelenler de "**esSesliler**" listesine eklenirler. Arama sırasında bu üç kök te aday olarak döndürülür.



Kök Düğümü (KokDugumu.java)

Kök düğümü sınıfı Kök ağacının yapıtaşıdır. Her düğümde aşağıda listelenen üyeler bulunur.

```
protected KokDugumu[] subNodes = null;
protected char harf;
protected List esSesliler = null;
protected Kok kok = null;
protected CharSequence kelime = null;
```

Kök Seçiciler

Kök Seçiciler temelde Kelime ağacı üzerinde belli kurallara göre yürüyerek kriterle uyan kökleri toplayan sınıflardır. Tüm kök seçiciler **KokSecici** arayüzünü gerçeklerler. Arayüzün yalnızca iki metodu vardır.

```
public interface KokSecici
{
    /**
     * @param giris: Uzerinde aday kok aramasi yapilacak giris kelimesi.
     * @return Aday kok dizisi
     */
    public List getAdayKokler(String giris);
}
```

Belli bir kelime için kök adayları ekde etmek istenirse getAdayKokler(String) metodu kullanılır. Bu metod kriterlere uyan tüm aday kökleri bir liste olarak döndürür.

KökSeciciFactory

Kök Seçiciler Sözlükten alınan bir fabrika yardımı ile elde edilirler. Aşağıdaki örnekte sozluk ismindeki TreeSozluk nesnesinden bir köksecici elde ediliyor.

```
KokSecici kokSecici = sozluk.getKokSeciciFactory().getKokSecici();
```

KökSeçici arayüzü:(**KokSecici.java**)

```
public interface KokSeciciFactory
{
    public KokSecici getKokSecici();
    public KokSecici getHataToleransliKoksecici();
    public KokSecici getTurkceHarfToleransliKokSecici();
}
```

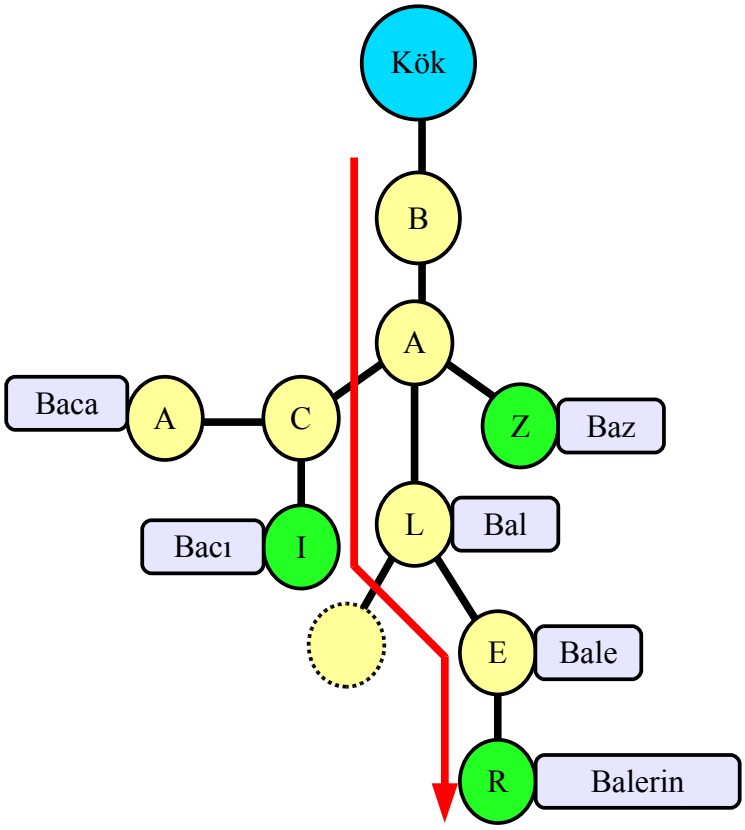
TreeSozluk.java'nın içinde bulunan KökSeçici Fabrikası gerçeklemesi (iç sınıf)

```
    class TreeKokSeciciFactoryImpl implements KokSeciciFactory
    {
        KokAgaci agac = null;
        public TreeKokSeciciFactoryImpl(KokAgaci agac)
        {
            this.agac = agac;
        }
        public KokSecici getKokSecici()
        {
            return new HizliWordTreeKokSecici(this.agac);
        }
        public KokSecici getHataToleransliKoksecici()
        {
            return new HataToleransliKokSecici(this.agac);
        }
        public KokSecici getTurkceHarfToleransliKokSecici()
        {
            return new TurkceHarfToleransliKokSecici(this.agac);
        }
    }
```

Normal Kök Seçici (HizliWordTreeKokSecici.java)

Çözümleyicinin verilen bir kelime için aday kökleri bulması için kullanılır. Giriş kelimesinin ilk harfinden başlanarak ağaç üzerinde ilerlenmeye başlanır. İlerleyecek yer kalmayana veya kelime bitene dek ağaç üzerinde ilerlenilir, ve rastlanan tüm kökler aday olarak toplanır. Örneğin “balerinlerin“ kelimesi için yolda rastlanan “bal”, “bale ve “balerin” aday

kökler olacaktır.

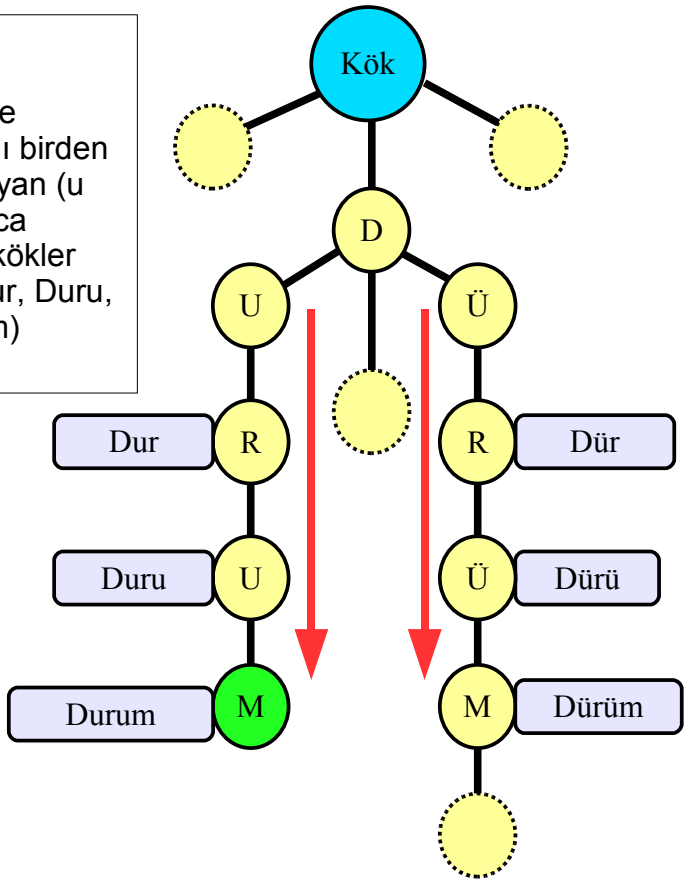


Türkçe Harf Toleranslı Kök Seçici (TurkceHarfToleransliKokSecici.java)

Bu seçici Deasciifier için kullanılır. Verilen kelime için ağaç üzerinde ilerlerken Türkçedeki karşılıkları birden fazla olabilecek harfler için (u-uü i-ıi o-oö vs.) alternatif dallarda da ilerlenerek yol üzerinde rastlanan tüm kökler toplanır

Giriş: **Durumunu**

Giriş kelimesinin harflerini ve harflerin Türkçedeki karşılığı birden fazla olabilecek harfleri taşıyan (u için u ve ü)düğümler boyunca ilerlenir. Ve rastlanan tüm kökler aday olarak döndürülür. (Dur, Duru, Durum, Dür, Dürü ve Dürüm)



Toleranslı Kök Seçici (ToleransliKokSecici.java)

Hatalı kelimeler için alternatif öneriler getirme işleminde kullanılır. Yapısı aynen Türkçe harf toleranslı seçici gibidir, ancak buradaki fark Türkçe'de dönüşebileceği harf alternatiflerine değil ağaç üzerinde ilerlerken oluşan kelime ile giriş kelimesi arasındaki duzeltmeMesafesinin (edit difference) belli bir değerden az olması (ön tanımlı değer :1) koşulunun gözetilmesidir.

Düzeltme mesafesinin ölçümü için Damerau Levenshtein **düzeltme mesafesi** (Edit Difference) ölçüm algoritması kullanılır. Ancak ağaç üzerinde ilerlerken kök adayları giriş kelimesinin sadece bir bölümünü oluşturacağından algoritma doğrudan uygulanmaz. Kök üzerinde ilerlenirken elde edilen kökler giriş kelimesinin yalnızca baş tarafından kısa bir bölümünü oluşturacağından **subStringEditDistance(s1, s2)** şeklinde bir metod tanımlanmıştır. Bu metodda s1 stringinin belli bir hata mesafesi ile s2 stringinin parçası olup olamayacağı kontrol edilir.

Örnek Kullanım:

```
...
// Binary Sozluk Okuyucu oluştur
SozlukOkuyucu sozlukOkuyucu = new BinarySozlukOkuyucu();
// Sözlük ile ilklendir
sozlukOkuyucu.initialize("kaynaklar/kb/binary-sozluk.bin");
// Ağaç sözlüğü oluştur
Sozluk sozluk = new TreeSozluk(sozlukOkuyucu);
// Kök Seçici
KokSecici kokSecici = sozluk.getKokSeciciFactory().getKokSecici();
// Hata toleranslı seçici
KokSecici tolerans = sozluk.getKokSeciciFactory().getHataToleransliKoksecici();
// Çözümleyici oluştur
KelimeCozumleyici cozumleyici = new Cozumleyici(kokSecici);
// Çözümle :)
boolean turkceMi = cozumleyici.cozumle("Merhaba");
...
```

Zemberek Façade kullanımı.

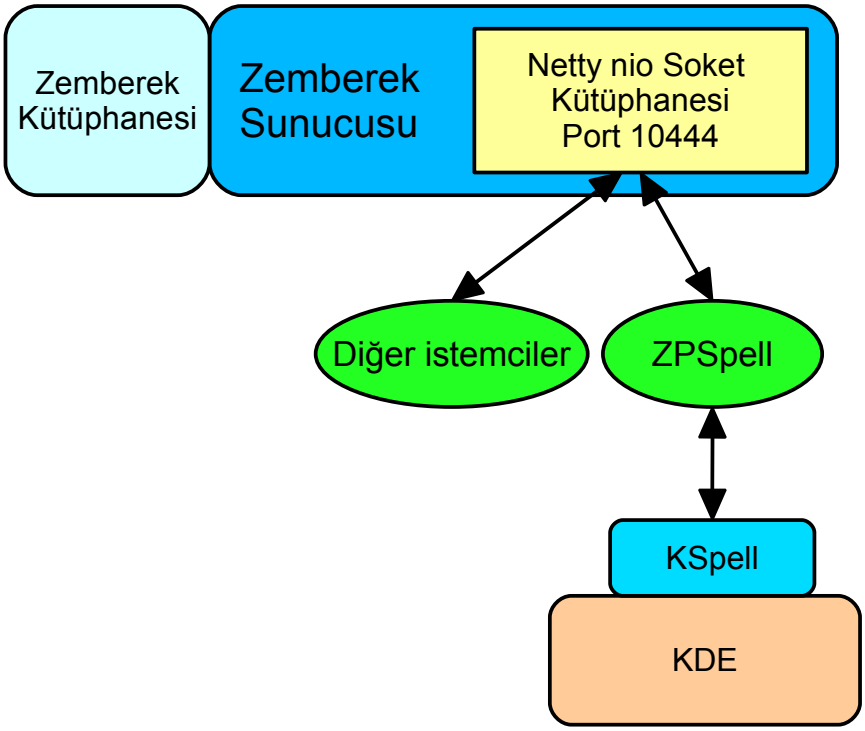
..... tamamla ...

Zemberek Sunucusu

Amaç

Uludağ-Pardus gibi Türkçe dil işlemlerine olan ihtiyacı yalnızca ofis yazılımları ile sınrlı olmayan büyük uygulamalarınher uygulama için ayrı bir Zemberek programı çalıştırması gereksiz bir yük getirecektir. Bu yüzden tüm uygulamaların paylaştığı merkezi bir türkçe NLP servisi fikrinden yola çıkılarak Zemberek sunucusu projesi başlatılmıştır.

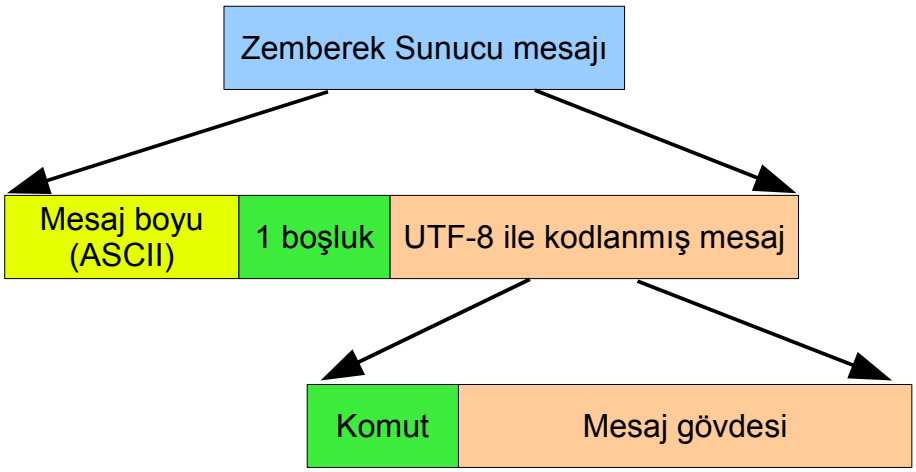
Ayrıca Java haricindeki dillerin zembereğin hizmetlerine erişebilmesi için platform ve dilden bağımsız bir arayüz ihtiyacı vardır. Aşağıdaki örnekde Zemberek-Pardus projesinde kullanılan yapı görülmektedir. Zemberek sunucusu 10444 nolu porttan istekleri dinler ve gerekli cevapları verir. Başka dillerdeki uygulamalar da bu basit protokolü gerçekleyerek Zemberek sunucusunun servislerinden faydalanabilirler.



Protokol

Protokolün yapısı son derece basittir. Tüm zemberek server protokol mesajları ASCII olarak kodlanmış mesaj uzunluğu bilgisi ile başlar. Bu uzunluk bilgisnin ardından bir boşluk ve UTF-8 ile kodlanmış mesajın kendisi gelir.Uzunluk bilgisine boşluk ve uzunluğun kendisi **dahil değildir,** sadece UTF-8 ile kodlanmış mesaj gövdesinin uzunluğu yazılır. Mesaj tipleri için şimdilik özel işaret karakterleri kullanılmıştır.

Mesaj karakteri	*istek*	*cevap*
*	Denetle	Kelime doğru
&	Öner	Öneriler
#		Kelime hatalı
?		Hata oluştu



Mesajlar

1. Denetleme mesajı: 9 * merhaba
2. Öneri mesajı: 8 & mrhaba
3. Denetleme sonucu, doğru ise: 1 *
4. Denetleme Sonucu yanlış ise 1 #
5. Öneri cevap mesajı: 28 & (halı,yalı,salı,çalı,malı)
6. Hata oluştu mesajı: 1 ?