2022-2023 EĞİTİM ÖĞRETİM YILI
GÜZ DÖNEMİ
BİL 423 - PROGRAMLAMA DİLLERİ DERSİ
DÖNEM SONU ÖDEVİ RAPORU
GRUP - PROGRAMLAMA DİLİ

19060381, Yasin Ünal
19060374, Kadir Emre Özer
19060912, Ahmed Furkan Kaymak
19061041, Osman Büyükşar

ARALIK 2022

Kısım I
Dili Tanıyalım

1 Scala'ya Giriş

Scala , programlama kalıplarını özlü, zarif ve güvenli bir şekilde ifade etmek için tasarlanmış, çok yönlü , ölçeklenebilir, modern ve çoklu paradigmaya sahip yüksek seviyeli bir statik programlama dilidir.

Martin Odersky tarafından geliştirilip 2004 yılında piyasaya sürüldü. Dilin adı, İngilizce "Scalable Language" olan "ölçeklenebilir dil" ifadesinden gelir. Ölçeklenebilirlik özelliği hibrit olmasındandır.

Java diline çok benzerdir.Kısa ve öz olacak şekilde tasarlanmıştır ve tasarım kararlarının çoğu Java eleştirilerini ele almaktır[wiki]. Farklı platformlarda çalışabilmesi , çoklu paradigmalı olması , Java ile birlikte çalışabilmesi , kolay okunabilirliği , sade sözdizimi , verimli kodlama imkanı , güçlü statik dil olması dili güçlü kılan özelliklerdendir. Scala, nesne yönelimli ve fonksiyonel dillerin özelliklerini sorunsuz bir şekilde bütünleştirir.

Farklı kullanım alanları vardır. Başlıca web ile mobil uygulamaları , büyük veri ve veri işleme uygulamaları , işletme yazılımları ve mobil uygulamalarda kullanılır.

Scala, Java platformu üzerinde çalışabilen bir dildir. Dalvik bayt koduna da çevrilebilir. Bu yüzden Android geliştirmeye uygundur.

Derleyici ve kütüphaneleri de dahil Apache lisansı altında yayınlanmıştır. [15] [16]

2 Dilin Tarihçesi

Martin Odersky daha öncesinde javac derleyicisi üzerinde çalışmıştı ve Funnel dili üzerinde çalışırken Scala'yı geliştirme fikri doğdu. 2001 yılında geliştirilmeye başlandı ve 2004 yılında Java platformunda yayınlandı.

2006 yılında ikinci sürümü yayınlandı. 2011 yılında Scala ekibi, Avrupa Araştırma Konseyi (ERC)'nden büyük bir araştırma hibesi aldı. Daha sonrasında Odersky ve arkadaşları Typesafe Inc. şirketini kurdu ve farklı yatırımlar alarak dili geliştirmeye devam etti. [17]

Şekil 1: Martin Odersky

3 Scala Dilinin Özellikleri

- Scala genel amaçlı kullanılan hibrit bir dildir. Yani hem fonksiyonel özellikleri barındıran , hem nesne yönelimli hem de emir esaslı bir dildir. Bu özellikleri esnek bir şekilde bir araya getirir.

- İşlevsel veya fonksiyonel bir dil olması : Yani işlevin birinci sınıf değerler olarak kullanılmasını destekler. Bu kodun nasıl yapması gerektiğinden çok ne yapması gerektiğine odaklanan daha bildirimsel bir programlama stili sağlar
- Nesne yönelimli olması : Sınıfların, özelliklerin ve nesnelerin tanımlanmasına izin verir. Scala'da herşey nesnedir. Gerçek dünya kavramlarını sezgisel bir şekilde modellemeyi mümkün kılar.
- Güçlü statik tiplemelidir (Sanılanın aksine dinamik değildir). Bu bir değişkenin türünün , derleme zamanında bilinmesi gerektiği anlamına gelir. Daha verimli kod sağlar ve geliştirme sürecinde ilgili hataların yakalanmasına yardımcı olabilir.
- Soyutlamaların tanımlanmasında esnektir.
- Okuması ve yazması kolay olan , sade ve çok yönlü bir sözdizimine sahiptir.
- Verimli kodlamaya izin verir. Hata yakalama için tür çıkarımı özelliği vardır.
- Eşzamanlı programlama desteğine sahiptir. Yukarıdan aşağıya yaklaşımı izler. Programların her biri birden fazla parçaya bölünerek işlenebilir. Bu sayede çok thread'li paralel programalanmış programlar yazılabilir.
- Test esilmesi ve yeniden kullanılması kolaydır.

[16]

Şekil 2: Scala'nın desteklediği platformlar

3.1 Java desteği

- Java dili ile ortak kullanılabilirlik özelliği vardır. Sorunsuz bir şekilde Java kodu ile etkileşime girebilir.
- Scala kaynak kodu Java bayt koduna derlenebilir. Ardından bir Java sanal makinesi'nde (JVM) çalıştırılabilir.
- Bu sayede Java kütüphanelerini ve framework'leri Scala içerisinde kullanılabilir.

3.2 Diğer Teknolojiler

- NET Framework'leri ile iyi çalışabilecek şekilde tasarlanmıştır.
- Bir tarayıcıda çalıştırılabilir. Çünkü Javascript'e derlenebilir.
- Hızlı derlenebilme için LLVM teknolojisini destekler. Bu

4 Hangi Alanlarda Kullanılır

Scala programlama dilinin kullanım alanları çeşitlidir ve dilin çok yönlülüğü nedeniyle farklı alanlarda kullanılabilir. Aşağıda, Scala programlama dilinin kullanım alanlarının bazıları verilmiştir:

1. Web uygulamaları: Java Virtual Machine (JVM) üzerinde çalışması nedeniyle, web uygulamaları geliştirme konusunda oldukça etkili bir dil olabilir. Scala programlama dilinin çok yönlülüğü, web uygulamalarında kullanılabilecek çeşitli özellikleri barındırmasına yardımcı olur. Örneğin, Scala programlama dilinde, sınıflar, nesneler ve traitler gibi özellikler kullanılabilir ve bu özellikler, web uygulamalarında etkili bir şekilde kullanılabilir.

2. Big Data işleme: Scala programlama dilinin, veri işleme konusunda oldukça etkili bir dil olması nedeniyle, Big Data işleme alanında da kullanılabilir. Özellikle, Scala programlama dilinin, Apache Spark gibi Big Data işleme platformları ile bütünleşik çalışması nedeniyle, Big Data işleme konusunda etkili bir dil olabilir. Scala programlama dilinin, veri işleme konusunda çok yönlülük sağlaması ve performansının yüksek olması nedeniyle, Big Data işleme alanında sıklıkla tercih edilir.

3. İşletme yazılımı: JVM üzerinde çalışması nedeniyle, işletme yazılımı alanında da kullanılabilir. Özellikle, Scala programlama dilinin, Java ile uyumlu çalışması nedeniyle, işletme yazılımı alanında tercih edilebilir. Çok yönlülüğü ve performansı nedeniyle avantaj sağlar.

4. Veri bilimi ve makine öğrenimi: Scala programlama dilinin, veri işleme konusunda etkili bir dil olması nedeniyle, veri bilimi ve makine öğrenimi alanlarında da kullanılabilir. Özellikle, Apache Spark gibi veri işleme platformları ile bütünleşik çalışması nedeniyle, veri bilimi ve makine öğrenimi alanlarında etkili bir dil olabilir. Veri işleme konusunda çok yönlülük sağlaması ve performansının yüksek olması nedeniyle, veri bilimi ve makine öğrenimi alanlarında sıklıkla tercih edilir.

5. Mobil uygulamalar: Dil, mobil uygulamalar geliştirme konusunda da etkili olabilir. Özellikle, Android Studio gibi mobil uygulama geliştirme ortamları ile bütünleşik çalışması nedeniyle, mobil uygulamalar geliştirme konusunda kullanılabilir.

[1]

5 Kütüphane ve Frameworkler

Scala programlama dilinde, çeşitli framework ve kütüphane mevcuttur. Aşağıda, Scala dilinde kullanılabilecek bazı önemli framework ve kütüphanelerin bir listesi verilmiştir:

1. Play Framework: Bu framework, çok yönlü ve etkileşimli web uygulamaları oluşturmak için kullanılabilir.

2. Akka: Bu framework, esnek ve ölçeklenebilir bir actor modeli tabanlı uygulama oluşturmak için kullanılabilir.

3. Scalaz: Bu kütüphane, Scala dilinde işlevsel programlama desteği sağlar.

4. ScalikeJDBC: Bu kütüphane, Scala dilinde JDBC tabanlı veritabanı erişimini kolaylaştırır.

5. Slick: Bu kütüphane, Scala dilinde veritabanı erişimini ve işlemeyi kolaylaştıran bir araçtır.

6. scala-async: Bu kütüphane, Scala dilinde asenkron programlama desteği sağlar.

7. scala-parser-combinators: Bu kütüphane, Scala dilinde parser oluşturmayı kolaylaştırır.

8. scala-swing: Bu kütüphane, Scala dilinde masaüstü uygulamaları oluşturmak için kullanılabilir.

Bu listede sadece bazı önemli Scala framework ve kütüphaneleri verilmiştir. Scala dilinin geniş bir ekosistemi vardır ve bu ekosistemde çok sayıda farklı çözüm mevcuttur.

Diğer yandan Scala dilinin çok önemli bir framework'ü daha vardır. Bu framework Spark'tır.

5.1 Spark

Apache Spark, distribütörel veri işleme motoru olarak kullanılan bir framework'tür. Spark, veri işleme, veri madenciliği ve veri analitiği gibi işlemler için kullanılır. Spark, veri işlemeyi hızlı bir şekilde gerçekleştirir ve büyük veri setlerini işlemeye uygundur.

Spark, çeşitli veri kaynaklarından veri toplama, veri temizleme, veri madenciliği ve veri analitiği gibi işlemleri gerçekleştirir. Spark, veri işleme işlemlerini yüksek performanslı bir şekilde gerçekleştirir ve bu sayede büyük veri setlerini işlemeye uygun bir araçtır.

Spark, çeşitli programlama dilleriyle (Java, Scala, Python gibi) kullanılabilir ve birçok çeşitli veri depolama sistemleriyle (Hadoop HDFS, Amazon S3 gibi) uyumludur. Spark, ayrıca birçok çeşitli veri depolama sistemlerine (relational veritabanları, NoSQL veritabanları gibi) erişim sağlar.

Spark, veri işleme işlemleri için kullanılabilecek çeşitli araçlar içerir. Bu araçlar arasında, SQL sorguları, veri madenciliği modelleri, stream işleme gibi araçlar bulunur.
Örneğin, veri işleme işlemlerinde çeşitli veri yapılarını (dataframe, dataset gibi) destekler.Veri işleme işlemlerinde farklı veri depolama sistemlerinden veri çekme ve veri yazma işlemlerini destekler.

Veri bütünlüğünü ve güvenliğini sağlamaya yönelik özellikler sunar. Veri değişikliklerini izleme ve veri geri yükleme gibi özellikler sunar.

İşlemlerinde paralel işleme özelliği sunar. Bu sayede, veri işleme işlemleri daha hızlı gerçekleştirilebilir ve büyük veri setleri daha hızlı işlenebilir.

Daha detaylı bilgi için, Spark'ın belgelerine ve öğreticilerine bakabilirsiniz. [14]

https://spark.apache.org/docs/latest/](https://spark.apache.org/docs/latest/ https://spark.apache.org/docs/latest/api/scala/org/apache/spark/index.html

6 Scala Dilinin Semantic Bilgileri

Scala dilinin parser'ı, Scala dilinin kodlarını, dilin kendi kurallarına göre yazılmış olan sözdizimi (syntax) kurallarına göre ayrıştırır ve bu ayrıştırma sonucu oluşan verileri, dilin özel bir anlamı olan semantik verilere dönüştürür. Bu semantik veriler, Scala dilinin kodlarının anlamını ifade eder ve bu veriler, bilgisayar tarafından anlaşılır bir şekilde saklanır ve işlenir.

1. Lexical Analiz: Dilin kodlarının dilin kendi kurallarına göre yazılmış olan sözdizimi (syntax) kurallarına göre ayrıştırılması sürecine verilen isimdir. Lexical analysis süreci, dilin kodlarını, dilin sözdizimi (syntax) kurallarına göre ayrıştırır ve bu ayrıştırma sonucu oluşan verileri, dilin özel bir anlamı olan semantik verilere dönüştürür.[6]
   Scala dilinde, lexical analiz süreci, Scala dilinin parser'ı tarafından yapılır. Scala parser'ı, dilin kodlarını dilin sözdizimi (syntax) kurallarına göre ayrıştırır ve bu ayrıştırma sonucu oluşan verileri, dilin özel bir anlamı olan semantik verilere dönüştürür. Bu sayede kodlar, dilin anlamını ifade eden semantik verilere dönüştürülür , bilgisayar tarafından anlaşılır ve çalışır hale gelir.

2. Semantik Analiz: Dilin anlamını ifade eden verileri inceleme sürecine verilen isimdir. Semantik analiz süreci, dilin kodlarının anlamını belirler ve bu verileri, dilin özel bir anlamı olan semantik verilere dönüştürür.

Oluşan birimler ise aşağıdakine benzer yapılardır.

- Lexeme: Lexeme (lexem), dilin sözdizimi (syntax) kurallarına göre ayrıştırılan dilin kodlarının en küçük anlamlı birimlerine verilen isimdir. Örneğin, Scala dilinde "val" ve "a" kelimeleri lexeme olarak kabul edilir.

- Token: Token, dilin lexeme'lerinin bir araya gelerek oluşturduğu dilin sözdizimi (syntax) kurallarına uygun birimlere verilen isimdir. Örneğin, Scala dilinde "val a= 10" ifadesi bir token olarak kabul edilir.

6.1 Scala Parser

Genellikle bir parser konusu açıldığında aklımızda ün kazanmış diyebileceğimiz yacc,Bison ya ANTLR gibi C ve Java'da yazılmış ayrıştırıcılar gelmektedir fakat bu ayrıştırıcılar kendilerine has programlama dillerini çalıştırmak için tasarlanmışlardır.Bu yaklaşım ayrıştırıcıların kullanım kapsamını kısıtlamaktadır.Bu nedenden dolayı Scala bu duruma eşsiz ve çözüm üreten bir alternatif sağlamaktadır.Ayrıştırıcı oluşturucunun bağımsız etki alanına özgü dilini kullanmak yerine(Standalone Domain Spesific Language) bir dahili etki alanına özgü dil kullananmaktadır(Dahili DSL).Dahili DSL ayrıştırıcılar için yapı taşı görevi görecek bir kitaplık olacaktır.[8]

Bir ayrıştırıcıyı(parser) G dilbilgisi için bir fonksiyona dökecek olursak:

$$parser : String => Option[Tree]$$

```
parser("12 + 10 * 2") = Some(12 ~ + ~ (10 ~ * ~ 2))
parser("12 + * 10 2") = None (i.e., syntax error)
```

Elimizde S sözcüğü olduğunu düşünürsek parser(S) işlevi G dilbilgisinden bu S sözcüğünü oluşturmak için bir türetme ağaçı üretecektir.Eğer ağaç türetilemezse geriye NONE dönüşü yapacaktır.

Scala'da ayrıştırma işlemi yapılırken birleştiriciler(combinators) kullanılır.Birleştiriciler basit fonksiyonları girdi olarak alan sonrasında onları birleştiren daha karmaşık fonksiyonlardır.

Scala, dilbilgisindeki her kuralı bir ayrıştırıcı işlevi olarak yorumlar ve EBNF işleçlerini, daha küçük ayrıştırıcı işlevlerini daha büyük işlevlerle birleştiren daha yüksek mertebeden işlevler (birleştiriciler olarak adlandırılır) olarak yorumlar.

6.2 Sums Of Prodcuts in Scala (SOP)

Daha kullanışlı bir ayrıştırıcı, ağaçları ifadeleri temsil eden sınıf örneklerine dönüştürmek için cases bloğunu kullanır. Bu ifadeler daha sonra yürütülebilir, assembly diline çevrilebilir, tür kontrol edilebilir, hata ayıklanabilir, yeniden düzenlenebilir vb.[9] Aşağıda aritmetik ifadelere özgü birkaç kural görülmektedir:

```
expr ::= term {"+" term | "-" term}.
term ::= factor {"*" factor | "/" factor}.
factor ::= ?FloatingPointNumber | "(" expr ")"
```

Bu ifadeleri bir birleştirici(combinator) ile Scala'da ifade etmek istersek:

```
import scala.util.parsing.combinator._
class Arith extends JavaTokenParsers
{
     def expr: Parser[Any] = term~rep("+"~term | "-"~term)
     def term: Parser[Any] = factor~rep("*"~factor | "/"~factor)
     def factor: Parser[Any] = floatingPointNumber | "("~expr~")"
}
```

Kullanıcıların rasgele sayı ürünleri toplamlarını ayrıştırmasına ve yürütmesine olanak tanıyan SOP yorumlayıcı oluşturacak olursak.

```
 -> 2
result = 2.0
-> 2 * 3
result = 6.0
-> 2 + 3
result = 5.0
-> 2 * 3 * 4 + 5 * 6 * 7
result = 234.0
-> 2 * 3 * (4 + 5) * 6 * 7
result = 2268.0
```

7 Scala vs Java

- Scala, statik olarak yazılan bir programlama dilidir, Java ise çok platformlu, ağ merkezli bir programlama dilidir.
- Scala, modern eşzamanlılığı desteklemek için bir aktör modeli kullanırken Java, eşzamanlılık için geleneksel iş parçacığı tabanlı modeli kullanır.
- Scala değişkenleri varsayılan olarak değişmez tiplerdir, Java değişkenleri ise varsayılan olarak değişken tiplerdir.
- Scala lazy evaluationı desteklerken Java desteklemez.
- Scala statik üyeler içermezken, Java statik üyeler içerir.
- Scala, operatörün aşırı yüklenmesini desteklerken Java, operatörün aşırı yüklenmesini desteklemez.
- Scala geriye dönük uyumluluk sunmazken, Java geriye dönük uyumluluk sunar.

- Scala geriye dönük uyumluluk sunmazken, Java geriye dönük uyumluluk sunar.
- İç içe kod nedeniyle Scala daha az okunabilirken, Java daha okunabilirdir.
- Scala framework Play ve Lift, Java framework ise Spring, Grails ve çok daha fazlasıdır.

Parametre	Scala	Java
Kod Karmaşıklığı	Göreceli olarak daha kısa	Nispeten daha büyük kod parçaları
Ne için tasarlandı	Hem nesne hem de işlevsel yönelimli bir dil olacak şekilde tasarlanmış ve geliştirilmiştir.	Nesne yönelimli olarak tasarlandı
Frameworks	Play , Spark , akka , scalding , neo4j	Spring, Grails ve çok daha fazlası
Tembel değerlendirme	Tembel değerlendirmeyi destekler	Tembel değerlendirmeyi desteklemiyor
statik üyeler	Statik üye yok	Statik üyeler içerir
Operatör Aşırı Yüklemesi	Destekler	Desteklemez
Derleme süreci	Derlenmesi nispeten yavaştır	Derlenmesi Scala'dan daha hızlıdır
URL yeniden yazma	Yeniden yazma gerekli	Yeniden yazmak gerekli değildir
Geriye dönük uyumluluk desteği	Scala geriye dönük uyumluluğu desteklemiyor	Java geriye dönük uyumluluğu destekler
Çoklu kalıtım desteği	Sınıfları kullanarak çoklu kalıtımı destekler, ancak soyut sınıfları desteklemez	Sınıfları kullanarak birden çok kalıtımı desteklemez, ancak arayüzlerle
Statik Anahtar Kelimesi	Static anahtar sözcüğünü içermez.	Statik anahtar sözcüğü içerir.
Değişken türü	Scala değişkenleri varsayılan olarak değişmez(val) tiptedir.	Java değişkenleri varsayılan olarak değiştirilebilir tiptedir.
Operatör Aşırı Yükleme	Operatör aşırı yüklemeye izin verir.	Operatörün aşırı yüklenmesini desteklemez.
Arama yöntemi	Scala'da, varlıklar üzerindeki tüm işlemler yöntem çağrıları kullanılarak gerçekleştirilir.	Operatörlere farklı davranılır ve çağrı yöntemi kullanılarak yapılmaz.
Okunabilirlik	Scala, iç içe kodu nedeniyle okunabilirliği azdır.	Java daha okunabilir.
Derleme süreci	Kaynak kodu sürecini bayt koduna derlemek çok yavaştır.	Kaynak kodu sürecini bayt koduna derlemek hızlıdır.

Tablo 1: Scala ve Java Arasındaki Farklar

8 Kurulum

Scala kurulumu için sbt aracının kurulması gerekir.

8.1 SBT - Standart Build Tool Kurulumu:

Öncelikle aracın aşağıdaki linkten indirilmesi gerekir.

http://www.scala-sbt.org/download.html

Kurulum tamamlandıktan sonra konsoldan 'sbt' komutu ile tamamlandığını görebiliriz. Burada 'sbt test' komutu ile gerekli uygulamaların kurulumu sağlanmalıdır. Bundan sonrasında kaynak kodlarımızı 'scalac' ve 'scala' komutları ile derleyip çalıştırabiliriz.

sbt için bazı komutlar aşağıdaki gibidir. [2]

Komut	Açıklama #
> help	Komut yardımı için kullanılır.
> new	Yeni proje oluşturur
> tasks	En yaygın kullanılan mevcut görevleri gösterir.5
> compile	Adım adım kod derlemek için kullanılır.
> test	Adım adım çalıştırıp test etmek için kullanılır.
> clean	Tüm yapıları siler.
> console	Scala REPL(Read, Eval, Print, Loop) çalıştırır.
> run	Derlenmiş class dosyasını çalıştırır.
> show x	Belirtilen değişkeni gösterir.
> eclipse	Eclipse projesi oluşturur.

Tablo 2: sbt komutları

8.2 Online Compiler

Scala programlama dilini denemek için kurulum yapmanıza gerek yoktur. Online compiler'lar aracılığı ile bu dili denenebilir.

Bu ödev kapsamında Scala resmi sitesi içerisinde de bulunan Scastie aracı kullanılacaktır. Bu araç ile herhangi indirme işlemi yapmadan scala programlanabilir.[2]

> https://scastie.scala-lang.org/

9 Scala Topluluk bilgileri

9.1 Forumlar

- users.scala-lang.org

Scala'da programlama hakkında sorular, tartışmalar ve duyurular için kullanılan forum, bu sitede her seviyeden geliştiriciler için soru-cevaplar mevcuttur. Forum kurallarına uyulduğu sürece her türlü soru cevap bulabilir. Aynı zamanda eski scala-user ve scala-announce gruplarının yerine geçer.

- contributors.scala-lang.org
Bu forum Scala'yı ileriye taşımakla ilgili konular için kullanılır. Scala Platform kitaplığı tartışmalarından Scala İyileştirme Süreci tartışmalarına, Scala derleyicisi, standart kitaplık ve modüller üzerindeki geliştirme çalışmalar gibi konular yer alır. Çekirdek bakımcılar ve açık kaynak katkıda bulunanların yanı sıra sırada neyin geldiğini görmek isteyenler ve dahil olmak isteyenlerde foruma katılabilir. Aynı zamanda eski scala-internals, scala-language, scala-debate, scala-sips ve scala-tools gruplarının yerine geçer.

- Teachers.scala-lang
Bu forumda Scala dilini öğretmek için Scala'nın kullanımına ilişkin tartışmalar yer alır. Scala dilini öğretmek için kullanılan malzemeleri, araçları ve yönergeleri barındırır.

9.2 Topluluk Kütüphaneleri ve Araçlar

- **index.scala-lang.org**
  Scala Center tarafından yürütülen Scala kitaplığı
- **index.scala-lang.org/awesome**
  Scala topluluğu tarafından yürütülen Scala diline ait framework ve araçlarla ilgili kitaplık
- **typelevel.org**
  newline Scala dili için uzantılar ve kütüphaneler sunar

9.3 Sosyal Medya ve Programlama Platformları

Scala programlama diline ait topluluklara aynı zamanda StackOverflow, github, reddit gibi medya platformlarından da ulaşmak mümkündür. Aynı zamanda twitter'da https://twitter.com/scala_lang adresinden haberlere de ulaşılabilir.

Kısım II
Kodlama

Scala dili de Java'da olduğu gibi derlenip , ardından oluşan kaynak kodun (byte kod) çalıştırılması ile çalışır.

Daha öncesinde sbt ile gerekenleri kurmuştuk. Şimdi programlamada adet olan hello world örneğini ilk olarak yapabiliriz.

Hello world

Basit bir Hello world uygulaması aşağıdaki gibidir.

```
/* Hello world uygulamasi */
object Hello {
def main(args: Array[String]) : Unit = {
    println("Hello, world")
    }
}


// Çıktı -> Hello, world!
```

Şimdi burada gördüğümüz yapıları açıklayalım

- Burada gördüğümüz object anahtar sözcüğü class ile benzer bir yapıdır. Ancak bu yapıyı sınıfın tek örneğini istediğimiz durumlarda kullanırız.
- main fonksiyonu ise Java'dan da hakim olduğumuz gibi statik bir metoddur. İçerisinde parametre olarak args değişkeni belirtilmiştir. Tür kullanımı ise args: Array[String] şeklinde yapılmaktadır. Yani içeriği String olan Listeleri argüman olarak kabul etmektedir.
- Fonksiyonlar'a dönüş tipi verilebilmektedir. Bunu fonksiyon belirtiminden sonra : *Int* şeklinde yaparız. Üstte gördüğümüz örnekte **Unit** anahtar kelimesi kullanılmıştır. Bu kelime fonksiyonun bir şey döndürmeyeceği durumlarda kullanılır. Bu gibi fonksiyonlara ise Scala'da Prosedür denmektedir.

Şimdi kodlarımızı çalıştırabiliriz. Bunu yapmak için öncelikle terminalden scalac komutu ile kaynak kodumuzu derlemeliyiz. Ardından ise scala komutu ile bu oluşan *.class* uzantılı dosyayı çalıştırmamız gerekir.

10 İsimlendirme Kuralları

Scala'da birtakım değişken isimlendirme kuralları bulunmaktadır.Bu kurallar aşağıda listelenmiştir:

- Değişken sembol olarak sadece _ ve $ içerebilir ayrıca $ ile başlayabilir.Diğer semboller(#,& vs.) ihlal oluşturacaktır.
- Değişken tanımında boşluk bulunamaz. var merha ba: Int = 10 şeklinde bir tanım yapılamaz.
- Unicode'u desteklediği için dile özgü harfler kullanılabilir ancak tavsiye edilmemektedir. "Ü Ğ I Ç "
- Değişken bir sayıyla başlayamaz.
- Değişken rezerve kelime içermemelidir.İçerse de bu bir ihlal değildir.

Dile ait rezerve kelimeler için. [10]

11 Kapsam

Scala'da değişkenler tanımlandıkları yere göre üç farklı kapsama sahiptirler.

11.1 Fields

Sınıf kapsamı içerisinde tanımlanmış değişkenlerdir.Sınıf içi metotların hepsi bu değişkenlere koşulsuz erişebilir.Field'lar sınıfın dışarısından da erişim belirtecine göre erişilebilir(getter).Bu tip değişkenler var veya val tiplemesiyle tanımlanabilir. Fields tipinde kapsama sahip değişkenleri aşağıdaki kod parçasında görülmektedir:

```
    class FieldScope
{
    var x = 10.3f // field
    var y = 7f // field
    def show() {
              println("Value of y : "+y);
        println("Value of x : "+x)
    }
}
```

11.2 Metot Parametreleri

Metota gönderilmiş olan parametrelerin kapsamıdır.Bu parametreler metot içerisinde kullanılabilir ancak metot parametresine işaret eden bir referans varsa bu değişken metot dışarısından da kullanılabilir. Aşağıda bu kapsama ait iki tane değişkenin tanımı verilmiştir.

```
    class MetotScope
{
```

```
    def multiply(s1: Int, s2: Int) // s1 ve s2
    {
        var result = s1 * s2
       println("Area is: " + result);
    }
}
```

11.3 Yerel Değişken

Metotların içerisinde tanımlanmış olan değişken türüdür.Sadece metotun içerisinden erişilebilir.var veya val tipinde olabilir.Bu kapsama ait iki tane değişkenin tanımı aşağıda yapılmıştır.

```
    class LocalVar
{
    def multiply()
    {
        var(s1, s2) = (3, 80); // s1 ve s2 değişkenleri
        var s = s1 * s2;
        println("Area is: " + s)
    }
}
```

12 Operatörler

12.1 Aritmetik Operatörler

Scala'da aritmetik işlemler için operatörler Tablo 3'de gösterilmektedir.

Operatör	Açıklama	Örnek
+	Topla	A + B
-	Çıkart	A - B
*	Çarp	A * B
/	Böl	B / A
%	Mod Al	B % A

Tablo 3: Scala'da Aritmetik Operatörler

12.2 İlişkisel Operatörler

Scala'da ilişkisel işlemler için operatörler Tablo 4'de gösterilmektedir.

Operatör	Açıklama	Örnek
==	Eşit	(A== B)
!=	Eşit Değil	(A != B)
>	Büyüktür	(A >B)
<	Küçüktür	(A <B)
>=	Büyük Eşit	(A >= B)
<=	Küçük Eşit	(A <= B)

Tablo 4: Scala'da İlişkisel Operatörler

12.3 Mantıksal Operatörler

Scala'da mantıksal operatörler Tablo 5'de gösterilmektedir.

Operatör	Açıklama	Örnek
&&	Mantıksal VE	(A && B)
||	Mantıksal VEYA	(A || B)
!	Mantıksal DEĞİL	!(A && B)

Tablo 5: Scala'da Mantıksal Operatörler

12.4 Bitsel Operatörler

Scala'da bitsel operatörler Tablo 6'de gösterilmektedir.Bu tablonun daha iyi anlaşılabilmesi iki değişken üzerinde çıktılarda paylaşılmıştır.Değişkenler:

```
A = 0011 1100 //60
B = 0000 1101 //13
```

Operatör	Açıklama	Örnek
&	Bitsel “VE”	(A & B) → 12 / 0000 1100
|	Bitsel "VEYA"	(A | B) → 61 / 0011 1101
^	Bitsel “XOR”	(A ^B) → 49 / 0011 0001
∼	Bitsel 1’e tümleyen	(∼A) → -61 / 1100 0011 (2’ye tümleyen)
<<	Bitsel sola kaydır	A <<2 → 240 / 1111 0000 (2 kere sola kaydır)
>>	Bitsel sağa kaydır	A >>2 → 15 / 1111(2 kere sağa kaydır)
>>>	Bitsel sağa kaydır,sıfırla doldur	A >>>2 → 15 / 0000 1111

Tablo 6: Scala Bitsel Operatörler

12.5 Atama Operatörleri

Scala’da bulunan atama operatörleri Tablo 7’de görülmektedir.

Operatör	Açıklama	Örnek
=	Temel atama operatörü	C= A + B
+=	Topla ve kendisine ata	C += A şuna denktir C= C + A
-=	Çıkar ve kendisine ata	C -= A şuna denktir C= C - A
*=	Çarp ve kendisine ata	C *= A şuna denktir C= C * A
/=	Böl ve kendisine ata	C /= A şuna denktir C= C / A
%=	Modunu al ve kendisine ata	C %= A şuna denktir C= C % A
<<=	Sola kaydır ve kendisine ata	C <<= 2 şuna denktir C= C <<2
>>=	Sağa kaydır ve kendisine ata	C >>= 2 şuna denktir C= C >>2
&=	Bitsel “AND” le ve kendisine ata	C &= 2 şuna denktir C= C & 2
^=	Bitsel “XOR” la ve kendisine ata	C ^= 2 şuna denktir C= C ^2
|=	Bitsel "OR"la ve kendisine ata	C |= 2 şuna denktir C= C | 2

Tablo 7: Scala’da Atama Operatörleri

12.6 Operatör Öncelik Sıraları

Scala'da operatörlerin öncelik sıraları Tablo 8'de görülmektedir.

Kategori	Operatör	Birleşim
Uç ek	() []	Soldan sağa
Tek işlenenli	! ~	Sağdan sola
Çarpma vb	* / %	Soldan sağa
Toplama	+ -	Soldan sağa
Kaydırma	>»>><<	Soldan sağa
İlişkisel	»= «=	Soldan sağa
Denklik,Eşitlik	== !=	Soldan sağa
Bitsel VE	&	Soldan sağa
Bitsel XOR	^	Soldan sağa
Bitsel VEYA	|	Soldan sağa
Mantıksal VE	&&	Soldan sağa
Mantıksal VEYA	||	Soldan sağa
Atama	= += -= *= /= %= >>= <<= &= ^= |=	Sağdan sola
Virgül	,	Soldan sağa

Tablo 8: Scala Operatör Öncelik Sırası

12.7 Tek İşlenenli Operatörler(Unary Operators)

Bir operatör işlevini yerine getirirken sadece bir tane işlenene veya değişkene ihtiyaç duyuyorsa bu operatör tek işlenenli olarak adlandırılır. Tablo 8'de de görüldüğü üzere "!" ve " " dışında tek işlenenli operatör bulunmamaktadır.

12.8 Üç İşlenenli Operatörler(Ternary Operators)

Bir operatör işlevini yerine getirirken üç tane işlenene veya değişkene ihtiyaç duyuyorsa bu operatör üç işlenenli olarak adlandırılır.Aşağıda Scala'daki tek üçlü operatörün kullanım örneği görülmektedir.

```
object Main extends App {
        var num1 = 100
        var num2 = 2
        var num3 = 200
        var num4 = 9000

        var large = 0

        large = if (num1 > num2) num1 else num2
```

```
        printf("Largest between %d and %d number is: %d\n", num1 ,
            num2, large)

        large = if (num3 > num4) num3 else num4
        printf("Largest between %d and %d number is: %d\n", num3 ,
            num4, large)
}
```

Scala geliştiricileri ne kadar bir operatör olarak görsede bu kullanımın bir operatör olarak kabul edilip edilmeyeceği Scala topluluklarında bir tartışma konusudur.

12.9 İki İşlenenli Operatörler(Binary Operators)

Bir operatör işlevini yerine getirirken iki tane işlenene veya değişkene ihtiyaç duyuyorsa bu operatör ik işlenenli olarak adlandırılır.Yukarıda bahsedilen tek işlenenli operatörler ve üç işlenenli operatörler haricindeki tüm operatörler iki işlenenlidir.

12.10 Operatör Aşırı Yükleme(Operator Overloading)

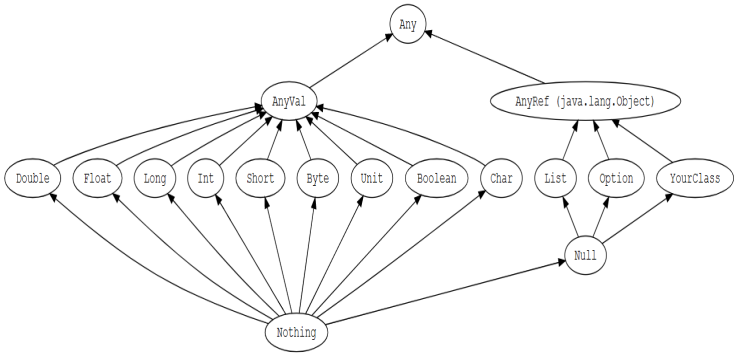
Scala operatör aşırı yüklemeye izin vermektedir.Bu özellik sayesinde var olan operatörler farklı işleri gerçekleştirmek(+ operatörü ile iki string'i toplamak gibi vs.) için kullanılabilir. Ne kadar operatör aşırı yükleme yazım hızını artırsa da tecrübeli olmayan yazılımcılar tarafından kullanıldığında kodun karmaşıklığını istenmeyen düzeyde artırabilmektedir.

```
class Complex(val real : Double, val imag : Double) {

  def +(that: Complex) =
           new Complex(this.real + that.real, this.imag + that.imag)

  def -(that: Complex) =
           new Complex(this.real - that.real, this.imag - that.imag)

  override def toString = real + " + " + imag + "i"

}

object Complex {
  def main(args : Array[String]) : Unit = {
       var a = new Complex(4.0,5.0)
       var b = new Complex(2.0,3.0)
       println(a) // 4.0 + 5.0i
       println(a + b) // 6.0 + 8.0i
       println(a - b) // 2.0 + 2.0i
  }
```

```
}
```

13 Veri Tipleri

Scala 'da tüm değerlerin bir veri tipi vardır. Bunu şekilde de görebiliriz. Veri tipleri birer tasarım deseni olarak düşünülebilir ve programlama dilinin temel bileşenlerindendir. Veri tipleri, bir programlama dilinde işlemlerin nasıl gerçekleştirileceğini ve verinin nasıl tutulacağını belirler. Scala programlama dilinde, veri tipleri iki ana kategoriye ayrılır: AnyVal ve AnyRef. Bu kategorilerin üzerinde ise Any vardır.[4]



Şekil 3: Scala'da tüm veri tipleri

Any tüm türlerin süper tipidir (supertype) ve aynı zamanda top-type olarak da adlandırılır. Equals , hashCode ve toString gibi metodları tanımlar.Any alt tip olarak Matchable tipine sahiptir. Bu tip desen eşleştirme yapmak için kullanılır.

Veri hiyerarşisi, Scala programlama dilinde veri tiplerinin nasıl birbirlerine ilişkili olduğunu gösterir. Örneğin, bir veri türü, başka bir veri türünden türetilebilir ve bu türetilen veri türü, türetilen veri türünün bir alt sınıfı olur. Bu, veri hiyerarşisinin temel öğesi olan miras alma ile sağlanır.

13.1 Tipleme Tipi(Dinamik-Statik)

Scala dilinde ne kadar sözdizimi devingen tipleme tarzında olsa da dil tamamıyla durağan tiplemelidir.Yazım sırasındaki bu devingen yazma tarzı derleme sırasında durağan tiplemeye çevrilmektedir.

13.2 Alt Türler

AnyVal , Scala programlama dilinde primitive veri tiplerinin tümünü kapsar. Programlama dilinin işleme kapasitesini arttırmak için tasarlanmıştır ve per-

formans açısından daha iyi çalışır. Örneğin, sayısal veri tipleri (int, double vb.), Boolean veri türü ve karakter veri türü (char) gibi.

AnyRef, Scala programlama dilinde, veri tiplerinin tümünü kapsar ve özel veri tiplerine (örneğin, sınıflar ve nesneler) ilişkin bir veri türüdür. Bu tipler referans tiplerdir. AnyRef veri tipleri, programlama dilinin işleme kapasitesini arttırmak için tasarlanmıştır ve performans açısından daha yavaş çalışır.

Scala programlama dilinde, Nothing ve Null, veri tipleri özel bir türüdür. Nothing, veri içermeyendir ve diğer tüm yapıların alt tipidir. Nothing hiçbir değerin dönmeyeceğini söyler.

Null, bir nesnenin olmadığını temsil eden bir veri türüdür ve bir nesne değişkenine atanırken kullanılır. Referans değerlerin alt-tipidir.

13.3 Unit

Unit hiçbir şey döndürmeyen fonksiyonlar için kullanılan veri tipidir. Bu fonksiyonlara prosedür de denmektedir. Bir bakıma void de denebilir.

13.4 Mantıksal Veri Tipleri

Scala'da tek tip mantıksal değişken bulunmaktadır.

```
val a: Boolean = true
val b: Boolean = false
```

Bu örnekte, "a" ve "b" değişkenleri boolean veri türlerine atanmıştır ve "a" değişkenine true (doğru) değeri, "b" değişkenine ise false (yanlış) değeri atanmıştır.

13.5 Aritmetik Veri Tipleri

Aritmetik veri türleri aşağıdaki örnekte olduğu gibi kullanılırlar.

```
val b: Byte = 1 //1 byte
val s: Short = 1 //2 byte
val i: Int = 1 //4 byte
val l: Long = 1 //8 byte

val c : Char = 'c' // 2 byte
val f: Float = 3.0 //32 bit
val d: Double = 2.0 //64 bit
```

Java dilinde de char tipi 2 byte büyüklüğündedir.C dilinde ise char 1 byte'lık yer işgal eder.Bunun sebebi C dili ASCII standartını Java Unicode'u destekler.Scala'da da char tipi 2 byte boyutundadır.

13.6 Char ve String

String'ler karakter katarlarıdır ve " " işaretleri (çift tırnak) arasında tanımlanabilirler. Char ise tek karakter kullanımı içindir. ' ' işaretleri (tek tırnak) arasında tanımlanırlar.

```
var symbol = '/'
var first_name = "Deacon"
var last_name = "John"
var middle_name = "St."

println(s"Name : $first_name $symbol$middle_name $last_name")
// Çıktı -> Name : Deacon /St. John
println(s"2 + 2 = ${2 + 2}")
// Çıktı -> 2 + 2 = 4
```

Eğer uzun bir string yazılmak isteniyorsa 3 tırnak işareti kullanılabilir.

```
    var quote = """ mesele en mutlu olduğun gün, en güzel hayaller
        kurduğun o gün ölmekmiş, mesele... neymiş mesele? mesele ö
        lmek değil,
    mesele dost bildiğin, en güvendiğin adamın eliyle ölmekmiş,
        mesele.."""
```

13.7 List ve Tuple

Liste oluşturmak için List anahtar kelimesi kullanılır. Listenin elemanlarının ne türde olacağı verilebilir.

```
    @main def main() =
    var liste = List(
    "kadir", // string
    2023,  // int
    'c',   // char
    true,  // bool
    () => "an anonymous function returning a string")
    println(liste)

// Çıktı -> List(kadir, 2023, c, true,
    main$package$$$Lambda$38787/0x0000000804d1dc10@4d1244ff)
```

Buradaki kod parçası çalıştırılabilirdir ancak tercih edilmemelidir.List yapısı kullanılırken genellikle tek bir veri tipi ile kullanılması istenir.Eğer değişik tarzda veriler depolanacaksa bunun için Tuple kullanılması önerilir.List yapısına dair birkaç metot incelemesi aşağıda sunulmuştur.

```
val numbers = List(10, 20, 30, 40, 10)
```

```
val list_in_list = List(List(1,2), List(3,4))

numbers.drop(2)                          // List(30, 40, 10)
numbers.dropWhile(_ < 25)                // List(30, 40, 10)
numbers.filter(_ < 25)                   // List(10, 20, 10)
numbers.slice(2,4)                       // List(30, 40)
numbers.tail                             // List(20, 30, 40, 10)
numbers.take(3)                          // List(10, 20, 30)
numbers.takeWhile(_ < 30)                // List(10, 20)
```

Tuple yapısına dair kod parçası aşağıda sunulmuştur.

```
val tuple3: (String, Int, Boolean) = ("Joe", 34, true)
val tuple4: (String, Int, Boolean, Char) = ("Joe", 34, true, 'A')
```

14 Tip Dönüşümleri

Scala programlama dilinde veri tipi dönüşümleri iki ana kategoriye ayrılır: otomatik veri tipi dönüşümleri ve özel veri tipi dönüşümleri.

Otomatik veri tipi dönüşümleri, programlama dilinin kendisi tarafından gerçekleştirilen veri tipi dönüşümleridir. Örneğin, Scala programlama dilinde int veri türünü double veri türüne otomatik olarak çevirir. Bu tip veri tipi dönüşümleri, programlama dilinin kendisi tarafından gerçekleştirildiği için, kod yazımı sırasında özel bir işlem gerektirmez.[13]

Özel veri tipi dönüşümleri ise, programlama dilinin kendisi tarafından gerçekleştirilmeyen veri tipi dönüşümleridir ve kod yazımı sırasında özel bir işlem gerektirir. Örneğin, Scala programlama dilinde int veri türünü string veri türüne çevirmek için, int veri türünü string veri türüne çeviren bir fonksiyon kullanılır. Özel veri tipi dönüşümleri, programlama dilinin farklı veri tiplerini birbirleriyle etkileşimde bulundurmak için sıklıkla kullanılır.

14.1 Otomatik Tip Dönüşümü

```
val intValue: Int = 10
val doubleValue: Double = intValue
```

intValue değeri otomatik olarak double veri türüne çevrildi.

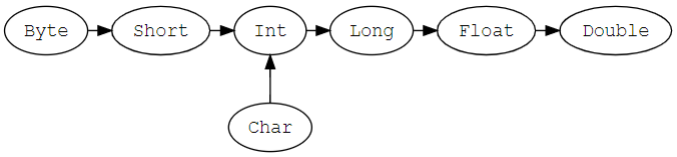
14.2 Özel Veri Tipi Dönüşümü

```
val intValue: Int = 10
val stringValue: String = intValue.toString()
val strValue: String = "10"
```

```
val intValues: Int = strValues.toInt()
```

Yukarıdaki örneklerde açık(explicit) olarak string-int dönüşümü örneği gösterilmiştir.

14.3 Temel Tip Dönüşüm Şablonu



Şekil 4: Scala'da Tip Dönüşüm Şablonu

Görüldüğü üzere sadece char tipindeki bir veri aritmetik veriye çevirilebilmektedir.Bunun sebebi her bir char değişkeninin bir Unicode ya da ASCII koduna denk gelmesindendir.Aşağıda basit bir char-int dönüşümü örneklenmiştir.

```
var ch : Char = 'A'
var char_casting: Int = ch
println(char_casting)
//Çıktı -> 65 (ASCII A = 0x65)
```

Bir byte-int dönüşümü:

```
val b: Byte = 127
val i: Int = b
```

15 Şartlı İfadeler

15.1 If-Else Bloğu

Scala dilinde basit bir if-else bölütü aşağıdaki gibi yazılabilir:

```
var x: Int = 3;

if(x == 3){
  println("x == 3")
}else if(x == 4){
  println("x == 4")
}else{
  println("x not found")
}
```

15.2 Match Bloğu

Scala dilinde basit bir switch-case döngüsü match anahtar sözcüğü ile gerçeklenir:

```
  val res = x match{
  case 1 => "one"
  case 2 => "two"
  case 3 => "three"
  case _ => "out of range" // varsayılan durum
}
println(res) // three
```

15.3 Try-Catch Bloğu

Scala'da bir try-catch-finally bloğu aşağıdaki gibidir.

```
try{
  val input = new FileReader("input.txt") // import
      java.io.FileReader
}catch{
  case ex:FileNotFoundException => println("The file doesn't
      exist in this directory") //import
      java.io.FileNotFoundException
}finally{
  println("Going out of the try-catch block")
}
```

16 Döngüler

16.1 For Döngüsü

Scala'da basit bir for döngüsü aşağıdaki gibi tanımlanabilmektedir.

```
for(i<- 0 to 100){ //100'ü de yazdıracak
  println(i)
}

for(i<- 0 until 100){ // 99'u yazdıracak (i<100)
  println(i)
}
var iterationList= List(3,2,1,4,4,6,7,8,9,11,2,3)
var j = 0; //burada j başlanacak indisi göstermemekte.
for(j <- iterationList){
  println(j)
}
```

```
// Çıktı -> 3,2,1,4,4,6,7,8,9,11,2,3
```

Bu örnekte, "i <- 1 to 10" ifadesi, "i" değişkeninin, 1 ile 10 arasındaki sayıları tek tek alacağı anlamına gelir.

16.2 while Döngüsü

while döngüsü ise C sözdizimi mantığında yapılabileceği gibi sonda do anahtar sözcüğü ile de yapılabilir.

```
var y = 33
while y < 44 do
  println(y)
  y+=1 // y++ söz dizimi desteklenmemektedir
```

17 Fonksiyonlar

Scala'da bir metot tanımı şöyle yapılmaktadır:

```
  def methodName(param1: Type1, param2: Type2): ReturnType =
```

Eğer bir metot geriye bir değer döndürmeyecekse (Java dilindeki void) geri dönüş değerine Unit yazılmadılır:

```
  def printIt(a: Any): Unit = println(a)
```

Scala'da basit bir metot uygulaması:

```
var int1 = 1
var int2 = 0xFF
var int3 = 777L
def sum(a: Int, b: Int = 10): Int = a + b
println(sum(int2,int3.toInt))
//Burada Long tipinde parametre Int tipine geçirilemez olacaktır.
//Çıktı ->1032 (777+255)
println(sum(int2)) //b varsayılan olarak 10 alınacaktır.

def combine_strings(s1: String, s2: String) = s1 + s2
println(combine_strings("sedat ","akleylek"))
// Çıktı -> sedat akleylek
```

Scala'da lambda fonksiyonu şu şekilde tanımlanabilir:

```
  var lamba_func = (str1:String, str2:String) => str1 + str2
```

```
var lamba_func_shorter = (_:String) + (_:String)
```

Buradaki iki lambda fonksiyonu da aynı işlevi gerçekleştirmektedir. Scala'da önemli farklardan birisi de fonksiyon ve metot ayrımıdır.Aşağıda aynı işi yapan fonksiyon ve metot verilmiştir.

```
def isEvenMethod(i: Int) = i % 2 == 0
val isEvenFunction = (i: Int) => i % 2 == 0
```

Burada fark edilebileceği üzere fonksiyonlar birer nesnedir.Yani bu demek oluyor ki bir fonksiyon List,Hash,Map gibi bir veri yapısı içerisinde tutulabilirken aynı işlemi metoda uygulamak hata verecektir çünkü metotlar bir sınıfa ait olmak zorundadır,tek başlarına var olamazlar.Scala 3 sürümü ile bu hata(eksiklik) giderilmiştir.Örtük olarak metotlar fonksiyonlara çevirilmektedir.

17.1 String Metodları

Scala dilinde, stringleri düzenlemek için birçok yararlı metod vardır. Aşağıda bazı örnekleri verilmiştir:

1. length : Bir stringin uzunluğunu döndürür.
2. concat : İki stringi birleştirir.
3. substring : Bir stringin belirli bir kısmını döndürür.
4. indexOf : Bir stringin içinde bir karakterin veya bir altstringin ilk oluşumunun indeksini döndürür.
5. toLowerCase : Bir stringi tüm küçük harflerden oluşan bir stringe çevirir.
6. toUpperCase : Bir stringi tüm büyük harflerden oluşan bir stringe çevirir.
7. trim : Bir stringin başındaki ve sonundaki boşlukları kaldırır.

Örnek kullanım:

```
val s = "Hello, world!"

println(s.length) // 13
println(s.concat(" How are you?")) // "Hello, world! How are
    you?"
println(s.substring(7, 12)) // "world"
println(s.indexOf('w')) // 7
println(s.toLowerCase()) // "hello, world!"
```

```
println(s.toUpperCase()) // "HELLO, WORLD!"
println("  Trim me!  ".trim()) // "Trim me!"
```

Kısım III
İleri Özellikler

18 Sınıf ve Nesne Yapısı

Scala aynı zamanda nesne yönelimli programlama paradigmalarına da sahiptir. Çoğu nesne yönelimli dil gibi sınıf ve nesne oluşturma da Scala dili içerisinde mevcuttur. Sınıfların ve nesnelerin kullanımı aşağıda ilgili başlıklarda daha detaylı incelenmiştir. Daha detaylı bilgiye kaynak[11]'ten ulaşabilirsiniz.

18.1 Sınf ve Nesne Oluşturma

Scala programlama dili nesne yönelimli özellikler barındırdığından, dil içerisinde sınıf ve nesne oluşturmaya izin veriyor.

Örnek olarak ismi Meyve olan bir sınıf aşağıdaki gibi tanımlanabilir.

```
class Meyve
val Mandalina = Meyve()
```

18.2 Sınıf Değişkenleri

Scala programlama dilinde sınıf değişkenleri varsayılan hali ile public olarak ayarlanıyor. Yani sınıf içerisindeki bir değişkene erişim belirteci yazılmadığında o değişken public erişime sahip oluyor. Aynı zamanda sınıf içerisindeki değişkenlere varsayılan değer atanabiliyor.

Aşağıdaki örnek ile bu durumu açıklayalım.

```
class Meyve():
  var renk = "Renk"
end Meyve

val Mandalina = Meyve()

println(Mandalina.renk)
Mandalina.renk = "Turuncu";

println(Mandalina.renk) //bu satır Turuncu çıktısını bastırır
```

18.3 Getter ve Setter Metod Tanımlama

Scala programlama dilinde sınıf değişkenlerini dışarıdan erişilemez hale getirmek için private erişim belirteci kullanabiliriz. Bu tür değişkenlere dışarıdan

erişmek için kullanıcağımız getter ve setter metodları aşağıdaki gibi tanımlanabilir.

```
class Meyve():
  var _renk = "Renk"

  /*renk değişkeni için getter metod*/
  def renk: String = _renk

  /*renk değişkeni için setter metod*/
  def renk_=(newValue: String): Unit =
        _renk = newValue

end Meyve


val Mandalina = Meyve()

Mandalina.renk = "kahverengi"

println(Mandalina.renk)
```

Yukarıdaki örnekte de görüldüğü gibi setter metod tanımlarken o fonksiyonu özel bir şekilde tanımlıyoruz. Setter metodlar tanımlamak için _= sembolünü kullanarak tanımlıyoruz bu sayede sadece fonksiyon ismi ve eşittir işareti kullanarak setter metodunu kullanabiliyoruz.

18.4 Constructor

Şimdiye kadarki örneklerde değişkenlerin değerini nesneyi oluşturduktan sonra tanımladık. Değişkenlerin değerini nesneyi oluştururken de tanımlayabiliriz. Aşağıda bunun için bir örnek verilmiştir.

```
class Meyve(var renk: String = "Renk")

  val Portakal = Meyve("Turuncu")
  println(Portakal.renk) //komut satırına Turuncu bastırır
```

Yukarıdaki örnekte görüldüğü gibi renk değişkenini nesneyi oluştururken tanımlayabiliyoruz. Eğer bunun gibi başka bir constructor tanımlamak istersek aşağıdaki gibi bir kod yazabiliriz.

```
class Meyve(var renk: String = "Renk"):

  var _x = 0
  def this(renk: String,sayi: Int) =
        this(renk)
```

```
        _x = sayi

end Meyve

val Portakal = Meyve("Turuncu")
println(Portakal.renk) //komut satırına Turuncu bastırır

val Mandalina = Meyve("Turuncu", 1)
println(Mandalina._x) //komut satırına 1 yazdırır
```

Yukarıdaki gibi başka constructor tanımlamak istersek this kelimesi ile başka bir constructor tanımlayabiliriz. Burada önemli olan nokta yeni oluşturulan constructor'ların this anahtar kelimesi ile önceki constructor'ı çağırması gerektiğidir.

18.5 Metod Aşırı Yükleme

Scala programlama dili aynı zamanda metod aşırı yüklemeye izin veriyor. Öncelikle basit bir örnekle sınıf içerisinde metod oluşturmayı görelim:

```
class Meyve():
  def elmalariSoydum(elmaSayisi: String): Unit =
        println(elmaSayisi + " elma soydum")

end Meyve

val Mandalina = Meyve()
Mandalina.elmalariSoydum("10") //10 elma soydum yazdırır
```

Sınıf içerisinde metod tanımlama yukarıdaki gibi gerçekleştirilebiliyor. Bu metodun başka şekilde de çalışmasını istersek aşağıdaki örnekteki gibi metodu aşırı yükleyebiliriz.

```
class Meyve():
  def elmalariSoydum(elmaSayisi: String): Unit =
        println(elmaSayisi + " elma soydum")

  def elmalariSoydum(elmaSayisi: Int): Unit =
        println(elmaSayisi.toString + " elma soydum")

end Meyve

val Mandalina = Meyve()
Mandalina.elmalariSoydum("10") //10 elma soydum yazdırır
Mandalina.elmalariSoydum(5) //5 elma soydum yazdırır
```

Sınıf içerisinde metod tanımlama yukarıdaki gibi gerçekleştirilebiliyor.

Bu metodun başka şekilde de çalışmasını istersek aşağıdaki örnekteki gibi metodu aşırı yükleyebiliriz.

```
class Meyve():
  def elmalariSoydum(elmaSayisi: String): Unit =
        println(elmaSayisi + " elma soydum")

  def elmalariSoydum(elmaSayisi: Int): Unit =
        println(elmaSayisi.toString + " elma soydum")

end Meyve

val Mandalina = Meyve()
Mandalina.elmalariSoydum("10") //10 elma soydum yazdırır
Mandalina.elmalariSoydum(5) //5 elma soydum yazdırır
```

19 Sınıflar Arası Miras Alma (Inheritance)

Scala programlama dili nesne yönelimli paradigmaya ait olan sınıflarda miras almayı destekliyor. Bu sayede polimorfizm, dynamic binding gibi nesne yönelimli paradigmaya ait bir çok özellik scala programlama dilinde gerçeklenebiliyor. Bu özellikler mixin, trait, abstract class veya sıradan sınıflar ile gerçeklenebiliyor. Aşağıdaki konu başlıklarında bunların kullanımları ve kısıtları detaylı bir şekilde anlatılacaktır.

19.1 Trait

Trait'ler sınıfların ortak özellikleri bir çatı altına toplanmak istendiğinde kullanılıyor. Aynı Java programlama dillerinde interface'e denk geliyor. Trait'lerde de aynı interface gibi nesne oluşturulamıyor ve parametre alınamıyor. Basit bir trait aşağıdaki gibi oluşturulabiliyor.

```
trait Tatlı
```

Bu kullandığımız trait içerisinde bir metod veya değişken yok, eğer bu gibi özellikleri eklemek istersek tanımlamayı aşağıdaki gibi değiştirebiliriz.

```
trait Tatli:
  val sekerMiktari: Int
  def tatliTuru(): String

class SutluTatli extends Tatli:
  val sekerMiktari: Int = 10
  override def tatliTuru(): String = "Sütlü"
```

```
end SutluTatli

class SerbetliTatli extends Tatli:
  val sekerMiktari: Int = 100
  override def tatliTuru(): String = "Şerbetli"

end SerbetliTatli

val Trilece = SutluTatli()
println(Trilece.tatliTuru())
```

Yukarıda gösterildiği gibi trait'lere fonksiyon veya değişken tanımlanabilir. Yazılacak sınıflar bu trait'leri extend edebilir. Ancak aynı Java dilindeki gibi tanımları yazılan metodların miras alan sınıf içerisinde deklare edilmesi gerekiyor.

Ayrıca trait'lerde Java dilindeki interface'den farklı olarak fonksiyonlar trait içerisinde deklare edilebiliyor. Aşağıdaki örnekle bu durumu gösterebiliriz.

```
 trait Tatli:
    val sekerMiktari: Int
    def tatliTuru(): String = "Tatli"


class SutluTatli extends Tatli:
    val sekerMiktari: Int = 10
    //override def tatliTuru(): String = "Sütlü"

end SutluTatli

class SerbetliTatli extends Tatli:
    val sekerMiktari: Int = 100
    override def tatliTuru(): String = "Şerbetli"

val Trilece = SutluTatli()
println(Trilece.tatliTuru()) // komut satırına Tatli yazdirir
```

Yukarıda da görüldüğü gibi Java dilinden farklı olarak Scala dilinde Trait'lerde metod tanımlarından farklı olarak deklarasyon da koyabiliyoruz. Ayrıca trait içerisinde deklare edilmiş metodların bu trait'i miras alan sınıf içerisinde override edilmesi şartı kalkıyor.Daha detalı bilgi için kaynak [12]'e bakabilirsiniz.

19.2 Subtyping

Scala dili üzerinde polimorfizm kullanılmak istendiğinde subtyping kullanılarak bu işlem gerçeklenebiliyor. Aşağıda scala dilinde polimorfizm için bir

örnek verilmiştir.

```
import scala.collection.mutable.ArrayBuffer

trait Tatli:
  val sekerMiktari: Int
  def tatliTuru(): String = "Tatlı"

class SutluTatli extends Tatli:
  val sekerMiktari: Int = 10
  override def tatliTuru(): String = "Sütlü"
end SutluTatli

class SerbetliTatli extends Tatli:
  val sekerMiktari: Int = 100
  override def tatliTuru(): String = "Şerbetli"
end SerbetliTatli

val Tatlilar = ArrayBuffer.empty[Tatli]
Tatlilar.append(SutluTatli())
Tatlilar.append(SerbetliTatli())

Tatlilar.foreach(tatlim => println(tatlim.tatliTuru()))
```

Yukarıda da görüldüğü gibi trait kullanılarak scala dilinde polimorfizm gerçeklenebilir.

19.3 Mixins

Scala programlama dilinde sınıf oluşturmak için kullanılan trait'lere Mixin'ler deniyor. Bir trait'in mixin olması için miras alınan trait'in içerisindeki metod veya değişkenlerin deklarasyonlarının da bulunması gerekiyor. Aşağıda mixin'lere bir örnek verilmiştir.

```
abstract class A:
  val message: String
class B extends A:
  val message = "Ben B sınıfının bir nesnesiyim"
trait C extends A:
  def loudMessage = message.toUpperCase()
class D extends B, C

val d = D()
println(d.message) // Ben B sınıfının bir nesnesiyim
println(d.loudMessage) // BEN B SINIFININ BİR NESNESİYİM
```

Buradaki D sınıfı hem B sınıfından hem de C mixin'inden miras alıyor. Detaylı bilgiye [7]'den ulaşabilirsiniz.

19.4 Inner Class

Scala dilinde, inner classlar bir sınıf içinde tanımlanan sınıflardır. Inner classlar, sadece içerisinde bulunduğu sınıf tarafından kullanılabilir ve ancak bu sınıfın nesnelerine özgüdür. Inner classlar, "class" anahtar sözcüğü ile tanımlanır ve sınıf adından sonra parantez içine veri üyelerinin ve işlevlerinin tanımı yer alır. Aşağıda inner class ile ilgili örnek verilmiştir

```
class SutluTatli{
    class Krema(değer: Int) {
      def değeriGöster(): Unit = {
        println(değer)
      }
    }
    val kremam = new Krema(5)
    kremam.değeriGöster() // 5
  }
```

19.5 Case Class

Scala dilinde, case classlar, veri üyelerinin tanımlı olduğu ve mutable olmayan (değiştirilemeyen) sınıflardır. Case classlar, "case class" anahtar sözcüğü ile tanımlanır ve sınıf adından sonra parantez içine veri üyelerinin tanımı yer alır. Case classlar, toString, equals ve hashCode gibi metotları otomatik olarak tanımlar ve kolayca nesne eşitliği kontrolü yapılmasını sağlar. Aşapıda case class ile ilgili örnek verilmiştir:

```
case class Öğrenci(isim: String, numara: Int)
val o1 = Öğrenci("Ali", 123)
val o2 = Öğrenci("Ali", 123)
val o3 = Öğrenci("Veli", 456)
println(o1 == o2) // true
println(o1 == o3) // false
```

19.6 Abstract Class

Scala dilinde, abstract classlar, miras alınabilecek ancak doğrudan nesne oluşturulamayacak sınıflardır. Abstract classlar, "abstract" anahtar sözcüğü ile tanımlanır ve içerisinde abstract veya concrete (tam olarak tanımlanmış) işlevler içerebilir. Abstract işlevler, sadece tanımı yapılmış ancak içeriği tanımlanmamış işlevlerdir ve alt sınıflar tarafından override edilmelidir. Aşağıda abstract class için örnek verilmiştir.

```
abstract class Canlı {
  def solunum(): Unit
```

```
  def yemekYe(): Unit
}

class Köpek extends Canlı {
  def solunum(): Unit = {
    println("Köpek soluyor")
  }
  def yemekYe(): Unit = {
    println("Köpek yemek yiyor")
  }
}
```

Abstract classlar, çeşitli amaçlarla kullanılabilir. Örneğin, bir sınıfın tüm alt sınıflarında ortak olan işlevleri tanımlayarak code duplication (kod çoğaltımı) önleyebilir veya bir sınıfın alt sınıfları tarafından zorunlu olarak implemente edilmesini istediğimiz işlevleri tanımlayabiliriz.

Scala dilinde, abstract classlar sadece abstract işlevler içerebilirler ve bu işlevler override edilmelidir. Ancak, concrete (tam olarak tanımlanmış) işlevler de içerebilirler ve bu işlevler override edilemez. Abstract classların veri üyeleri de tanımlanabilir ancak bu veri üyeleri doğrudan değiştirilemez, sadece getter ve setter metotları ile erişilebilir.

20 Scala Dilinde Fonksiyonel Özellikler

20.1 Higher-order functions (yüksek sıralı işlevler)

Scala dilinde, işlevler diğer işlevlerin argümanı olarak veya bir işlev döndüren işlevler olarak kullanılabilir. Bu sayede, işlevler diğer işlevlerin içinde kullanılabilir veya bir işlev olarak döndürülebilir. Aşağıda higher-order functions için örnek verilmiştir. Detaylı bilgi:[11]

```
def ikiKati(f: Int => Int, x: Int): Int = f(x) * 2
def kareEt(x: Int): Int = x * x

val sonuç1 = ikiKati(kareEt, 3) // 18
val sonuç2 = ikiKati((x: Int) => x + 1, 4) // 10
```

20.2 Closures (kapamalar)

Scala dilinde, işlevler dışarıdaki değişkenlere de erişebilir ve bu değişkenleri kapatarak saklar. Bu sayede, işlevler çağırıldıklarında dışarıdaki değişkenlerin değerlerini korur. Aşağıda closures için örnek verilmiştir.

```
var y = 5
def f(x: Int): Int = x + y
```

```
val sonuç = f(3) // 8
y = 7
val sonuç2 = f(3) // 10
```

20.3 Partial function application (kısmi işlev uygulaması)

Scala dilinde, işlevlerin bir kısmını önceden belirtilerek yeni bir işlev oluşturulabilir. Bu sayede, aynı işlevin farklı argümanlarla çağırılması için aynı kod tekrarı yapılmaz. Aşağıda partial function application için örnek verilmiştir.

```
def topla(x: Int, y: Int, z: Int): Int = x + y + z
val topla3 = topla(3, _: Int, _: Int)
val sonuç = topla3(4, 5) // 12
```

20.4 Çoklu Parametre Listesi (Currying)

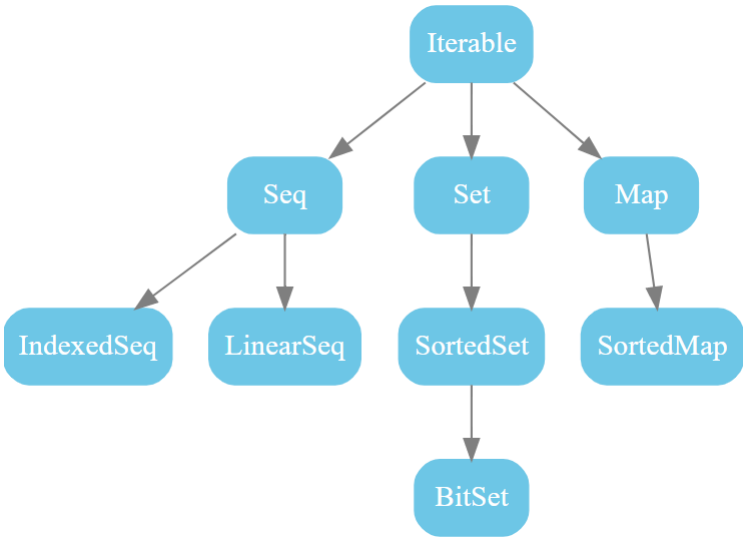
Scala dilinde, işlevler birden fazla argüman alabilir ve bu argümanların bir kısmını önceden belirtilerek yeni bir işlev oluşturulabilir. Bu sayede, aynı işlevin farklı argümanlarla çağırılması için aynı kod tekrarı yapılmaz.

21 İleri Seviye Veri Türleri

21.1 Collections

Scala programlama dilinde verileri tutmak için daha gelişmiş, farklı durum ve senaryolar için geliştirilen collections veri yapıları vardır. Bu Collections veri türleri 3 kategoride incelenir:

- **Sequences**
  Sıralı elemanları tutmak için kullanılan veri yapıları
- **Maps**
  Anahtar - değer çiftleri şeklinde bulunan elemenaları tutmak için kullanılan veri yapıları
- **Sets**
  Herhangi sıraya veya anahtar - değer çiftine sahip olmayan elemanları tutmak için kullanılan veri yapıları



Şekil 5: Collection hiyerarşisi

Şekil 5te görüldüğü gibi Collection veri yapıları Iterable ismindeki soyut bir sınıftan türetilmiştir. Ardından Sequence Set ve Map türleri kendi özelliklerine göre ayrılmaktadır. Daha detaylı bilgi için [3].

21.2 Sık Kullanılan Collectionlar

Scala programlama dilinde sık kullanılan collection yapıları tablo 9'te verilmiştir. Detaylı bilgi [3]

Collections	Immutable	Mutable	Açıklama
List	+		Lineer bağlı liste
Vector	+		İndekslenmiş sıralı liste
LazyList	+		Elemanları sadece ihtiyaç olduğunda hesaplanır.
ArrayBuffer		+	İndekslenmiş sıralı ve değiştirilebilen liste
ListBuffer		+	Değiştirilebilen bir List istendiğinde kullanılır.
Map	+	+	İndeksli anahter - değer çiftleri
Set	+	+	İndeksli kopya eleman içermeyen collection

Tablo 9: Sık kullanılan collections türleri

22 Hata Yakalama Yapıları

Scala dilinde hata yakalama yapıları, kodunuzda beklenmedik durumlarla karşılaştığınızda kodunuzun çalışmasını sürdürebilmesini sağlar. Bu yapılar, kodda bir hata gerçekleştiğinde programın akışının bozulmadan devam edecek şekilde programlanmasına olanak sağlar. Detaylı bilgi:[11]

22.1 Try-Catch Yapısı

Try-Catch yapısı, kodunuzda bir hata oluştuğunda hata mesajını yakalayarak işleme sokmanızı sağlar. Örnek olarak:

```
try {
  // Kodunuz burada
} catch {
  case e: Exception => // Hata mesajını işleme sokun
}
```

Try bloğu içerisinde yer alan kodunuz, hata oluştuğunda Catch bloğu içerisinde yer alan işlemleri gerçekleştirir. Bu sayede, hata oluştuğunda programın akışını devam ettirebilirsiniz.

22.2 Option Yapısı

Option yapısı, bir değişkende bir değerin olup olmadığını kontrol etmek için kullanılır. Örneğin:

```
val x: Option[Int] = Some(5)
x match {
  case Some(value) => // Değer varsa işlemlerinizi gerçekleştirin
  case None => // Değer yoksa işlemlerinizi gerçekleştirin
}
```

Detaylı bilgi için kaynak[5]'e bakabilirsiniz.

22.3 Either Yapısı

Either yapısı, bir değişkende iki farklı türde değer olabilme özelliğine sahiptir. Örneğin:

```
val x: Either[String, Int] = Right(5)
x match {
case Left(value) => // Değer String türündeyse işlemlerinizi
    gerçekleştirin
case Right(value) => // Değer Int türündeyse işlemlerinizi
    gerçekleştirin
}
```

Either yapısı, bir değişkende iki farklı türde değer olabilme özelliğine sahiptir. Bu sayede, değişkeninizde hangi türde değer olduğunu kontrol ederek işlemlerinizi gerçekleştirebilirsiniz.

Kaynaklar

[1] Meeting with scala. https://www.buraksenyurt.com/post/Scala-ile-TanÄśsmak.

[2] Installation of scala. https://firateski.com/ders-2-scala-ortam-kurulumu.

[3] Collections in scala. https://docs.scala-lang.org/scala3/book/collections-classes.html. ok.

[4] First look at the scala variables. https://docs.scala-lang.org/scala3/book/first-look-at-types.html.

[5] Functional exception handling in scala. https://docs.scala-lang.org/scala3/book/fp-functional-error-handling.html.

[6] Lexical analysis in scala. https://www.scala-lang.org/files/archive/spec/2.11/13-syntax-summary.html#lexical-syntax.

[7] Mixin and composition in scala. https://docs.scala-lang.org/tour/mixin-class-composition.html.

[8] Parsing in scala. http://www.cs.sjsu.edu/~pearce/modules/lectures/scala/parsing/\%E2\%80\%8Cindex.htm.

[9] Parsing in scala and sop combinator. http://www.cs.sjsu.edu/~pearce/modules/lectures/scala/parsing/\%E2\%80\%8Cindex.htm.

[10] Reserved words in scala. https://www.oreilly.com/library/view/learning-scala/9781449368814/apa.html.

[11] Scala documntation. https://docs.scala-lang.org/.

[12] Traits in scala. https://docs.scala-lang.org/tour/traits.html.

[13] Type casting in scala. https://docs.scala-lang.org/tour/unified-types.html.

[14] Spark. https://spark.apache.org/docs/latest/.

[15] Scala programming. https://en.wikipedia.org/wiki/Scala_(programming_language).

[16] Tour of scala. https://docs.scala-lang.org/tour/tour-of-scala.html.

[17] History of scala language. https://en.wikipedia.org/wiki/Scala_(programming_language)#History.