# Programlama Dilleri Projesi - Real-Time Grammar-Based Syntax Highlighter with GUI

BURSA TEKNİK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

**Muhammet Uğur Yaman**

**22360859023**

## 1. Projenin Amacı ve Kapsamı

Bu proje, kullanıcının gerçek zamanlı olarak bir kod editöründe programlama dili benzeri ifadeler yazarken sözdizimini denetleyen ve renklendirme (syntax highlighting) yapan bir uygulama geliştirmeyi hedefler. Temel amaç, kullanıcı herhangi bir satırı yazarken:  
- Leksiksel analiz (tokenlaştırma) yoluyla metindeki parçaları (anahtar kelime, sayı, operatör, tanımlayıcı vb.) tanımlamak,  
- Sözdizimi analizi (parser) kullanarak, tanımlı gramer kurallarına uygun olup olmadığını kontrol etmek,  
- Ekranda en az 5 farklı token tipini farklı renklerle gerçek zamanlı olarak vurgulamak,  
- Hata algılandığında o anki hatalı bölümü kırmızı altı çizili (underline) ve kenarına “❌ Hatalı Sözdizimi” uyarısını göstermek,  
- Geçerli kod yazımı durumunda ise pencerenin başlığında “✅ Kod Geçerli” şeklinde geri bildirim sunmaktır.  
  
Projenin teslim formatı üç aşamadan oluşur:  
1. Kaynak Kod (Source Code): Leksik analizci (lexer), sözdizimi analizci (parser), gerçek zamanlı renklendirme mekanizması ve GUI.  
2. Dökümantasyon (Documentation): Word veya PDF formatında “Language and Grammar Choice”, “Syntax Analysis Process”, “Lexical Analysis Details”, “Parsing Methodology”, “Highlighting Scheme” ve “GUI Implementation” başlıklarını içeren rapor.  
3. Demo (Makale ve Video): Proje çözümünü anlatan bir çevrimiçi makale ile uygulamanın çalışmasını gösteren bir video.  
  
Bu rapor, projenin final dökümantasyonu olarak hazırlanmıştır ve her bir başlığın altında kullanılan yöntemleri, tasarım kararlarını ve karşılaşılan zorlukları ayrıntılı olarak anlatır.

## 2. Programlama Dili ve Gramer Seçimi

### 2.1. Programlama Dili Seçimi

Bu projede Java dili tercih edilmiştir. Bu seçim şu nedenlere dayanmaktadır:  
- Swing kütüphanesi: Java’nın standart GUI çerçevesi Swing, JTextPane gibi zengin metin bileşenleri sunar. Bu bileşen, StyledDocument desteği sayesinde her tür metin stilini (renk, yazı tipi, altı çizili vb.) kolayca uygulamaya elverişlidir.  
- Regex (Düzenli İfadeler): Java’nın java.util.regex paketi, güçlü ve esnek düzenli ifade desteği sağlar. Leksik analiz için birden fazla token tipini aynı anda tanıyan “master regex” yapısını kolaylıkla kullanabildik.  
- Taşınabilirlik: Java platform bağımsız bir dildir; sonuçta ortaya çıkan JAR dosyası farklı işletim sistemlerinde (Windows, macOS, Linux) çalışabilir.  
- Öğrenme Kolaylığı: Proje kapsamında kullanılacak temel veri yapıları ve koleksiyonlar (List, Map vb.) Java’da yaygın ve anlaşılırdır. Bellek yönetimi (garbage collector) otomatik olduğundan, GUI ve analiz koduna odaklanmak daha kolaydır.

### 2.2. Gramer Seçimi

Projede, basit bir C/Java altkümesi (subset) grameri uygulanmıştır. Amaç, majör programlama dilleriyle uyumlu temel yapıların (değişken tanımlama, atama, if bloğu) gerçek zamanlı olarak analiz edilebilmesini sağlamaktır.  
  
Kullanılan Gramer (EBNF benzeri gösterim):  
  
<program> → (<statement>)\*  
  
<statement> → 'int' <identifier> '=' <number> ';'  
 | <identifier> '=' <identifier> <binary\_op> <number> ';'  
 | <identifier> '=' <signed\_number> ';'  
 | 'if' '(' <identifier> <rel\_op> <number> ')' '{' '}'  
  
<identifier> → [a-zA-Z\_][a-zA-Z0-9\_]\*  
  
<number> → [0-9]+  
  
<signed\_number> → ('+' | '-') <number>  
  
<binary\_op> → '+' | '-' | '\*' | '/'  
  
<rel\_op> → '>' | '<' | '==' | '!='  
  
<keyword> → 'int' | 'if'  
  
<operator> → <binary\_op> | '='  
  
<separator> → ';'  
  
<paren> → '(' | ')' | '{' | '}'  
  
Desteklenen Yapılar:  
1. Değişken Tanımlama:  
 int x = 5;  
 int sayı1 = 42;  
2. Atama (Assignment):  
 x = x + 6;  
 x = +6;  
 x = -3;  
 x = 6;  
3. If Bloğu (Boş İçerikli):  
 if(x > 3) { }  
 if(id < 10) { }  
  
Grammar’ın Kısıtlamaları:  
- Döngü (while, for), else, return vb. yapılar eklenmemiştir.  
- Fonksiyon tanımlama, parametreler, tip denetimi gibi gelişmiş konular ele alınmamıştır.

## 3. Sözdizimi (Syntax) Analiz Süreci

### 3.1. Parser Yaklaşım Tarzı

Projede Top-Down (predictive) yaklaşımına sahip, elle yazılmış bir parser kullanılmıştır. Özel olarak recursive-descent parser değil; bunun yerine satır ve token dizisi üzerine kural bazlı ilerleyen, el ile yazılmış bir yapı tercih edilmiştir. Her bir ifade (statement) tipi, basit bir if–else if zinciriyle denetlenir.

### 3.2. Satır ve Token Tabanlı Okuma

1. Token Listesi: İlk aşamada, girdi metni (fullText) Lexer.tokenize(fullText) çağrısıyla ayrıştırılır ve List<Lexer.Token> döner.  
2. İndeks Tabanlı İlerleme: Parser, int i = 0; while (i < tokens.size()) { … } döngüsü ile tokens.get(i) üzerinden sırayla tokenları okur.  
3. Kural Uygulaması:   
 - 'int' tanımlama,  
 - 'if' bloğu,  
 - Atama satırları,  
 - Diğer durumlar 'hata' şeklinde ele alınır.

### 3.3. Hata Tespiti ve Konum Belirleme

Parser, her kuralda beklenen token’ı bulamaz veya içerik eşleşmezse, hatalı token’ın start/end indeksleri errorStart/errorEnd değişkenlerine atanır ve false döndürür. GUI tarafı, bu alanları okuyup kırmızı altı çizili vurguyu uygular. IndexOutOfBounds hatalarını önlemek için her tokens.get(i + k) öncesi i + k < tokens.size() kontrolü yapılır.

## 4. Leksik (Lexical) Analiz Detayları

Leksik analiz (tokenizasyon) aşamasında master regex (tek bir düzenli ifade) kullanıldı. Java’nın java.util.regex paketi ile named-group (anahtar grup) tabanlı bir yapı oluşturuldu. Bu sayede token çakışmaları (örneğin 'int' hem KEYWORD hem IDENTIFIER olabilmesi) named-group sırası ile çözüldü. Örnek MASTER\_PATTERN:  
  
(?<KEYWORD>\b(?:int|if|else|while|return)\b)  
|(?<NUMBER>\b\d+\b)  
|(?<OPERATOR>[=+\-\*/<>!])  
|(?<PAREN>[(){}])  
|(?<SEPARATOR>;)  
|(?<IDENTIFIER>\b[a-zA-Z\_][a-zA-Z0-9\_]\*\b)  
  
Matcher.find() döngüsü ile eşleşen grup belirlenerek uygun TokenType oluşturuldu ve List<Token> olarak döndürüldü.

## 5. Parser (Sözdizimi Analizörü) Tasarımı ve Yöntemi

### 5.1. Genel Yaklaşım

Parser, elle yazılmış top-down yaklaşımıyla, token listesi üzerinde if–else if zinciri kullanarak gramer kurallarını denetler. Döngü içinde i indeksi ile tokens.get(i) okunur ve her kuralda i+k < tokens.size() kontrolleri yapılır.

### 5.2. Hata Tespiti ve Konumlandırma

Parser’da beklenen token gelmezse, o token’ın start/end indeksleri errorStart/errorEnd değişkenlerine atanır ve false döner. GUI, bu indeksleri okuyarak ilgili bölüme kırmızı alt çizili stil uygular.

### 5.3. Parser Kod Yapısı

Parser’ın ana akışı:   
- ‘int x = 5;’ kontrolü,   
- ‘if(x > 3){ }’ kontrolü,   
- ‘x = x + 6;’, ‘x = +6;’, ‘x = 6;’ atama kuralları,   
- Aksi hâlde hata.

### 5.4. Örnek Akışlar

1. Geçerli “int x = 5;”: token listesi doğru sıralanır, i += 5, sonrasında hata yok.  
2. Geçerli “x = x + 6;”: i += 6, döngü biter, hata yok.  
3. Geçerli “x = +6;”: i += 5, hata yok.  
4. Hatalı “int 2 = 5;”: token #1 NUMBER "2", hata yakalanır, errorStart=…  
5. Hatalı “x = x + ;”: noktalı virgül eksik, errorStart sayı token’ı, hata.  
6. Hatalı “if(x>10){”: kapanan ‘}’ eksik, errorStart “{”, hata.

## 6. Renklendirme (Highlighting) Şeması

Uygulamada aşağıdaki token tipleri ve renkler kullanıldı:  
1. KEYWORD: Mavi (int, if)  
2. NUMBER: Kırmızı (0, 5, 10, 12345)  
3. OPERATOR: Turuncu (=, +, -, \*, /, >, <, !)  
4. PAREN: Yeşil ((, ), {, })  
5. SEPARATOR: Siyah (;)  
6. IDENTIFIER: Siyah (x, sayı, \_temp)  
7. ERROR: Kırmızı altı çizili (hatalı token)  
  
Gerçek zamanlı performans optimizasyonu için DocumentListener + Timer (100 ms) kullanıldı. invokeLater ile GUI thread’de güvenli güncelleme yapıldı.

## 7. GUI (Grafiksel Kullanıcı Arayüzü) Uygulaması

### 7.1. Ana Bileşenler ve Yapı

Main.java: SwingUtilities.invokeLater ile SyntaxHighlighterGUI başlatılır.  
  
SyntaxHighlighterGUI.java:  
- JFrame, JTextPane, JScrollPane kullanılarak pencere oluşturulur.  
- Stiller setupStyles() ile tanımlanır (DEFAULT, KEYWORD, NUMBER, OPERATOR, PAREN, SEPARATOR, IDENTIFIER, ERROR).  
- 100 ms gecikmeli Timer ile highlight() metodu çağrılır.  
- DocumentListener ile her metin değişiminde Timer.restart() edilir.  
- highlight():   
 1. Tüm metni DEFAULT (siyah) ile sıfırla  
 2. Lexer.tokenize(text) → token listesi → her token için renk uygula  
 3. Parser.parse(...) → hata var mı kontrol et → hata varsa errorStart/errorEnd ile ERROR stili uygula  
 4. Başlık “✅ Kod Geçerli” veya “❌ Hatalı Sözdizimi” olarak güncellenir.

## 8. Karşılaşılan Zorluklar ve Çözümler

1. Token Çakışmaları:   
 - 'int' hem KEYWORD hem IDENTIFIER olabilir → master regex içinde KEYWORD en üstte yer aldı.  
  
2. IndexOutOfBoundsException Hataları:  
 - Parser’ın tokens.get(i + k) çağrılarından önce i + k < tokens.size() kontrolü yapıldı.  
 - Eksik token durumunda, bir önceki token’ın start/end indeksleri errorStart/errorEnd olarak atandı.  
  
3. Gerçek Zamanlı Renklendirme Performansı:  
 - Her tuşta renklendirme GUI’yi yavaşlatıyordu → 100 ms gecikmeli Timer kullanıldı.  
 - invokeLater ile GUI thread’de güvenli güncelleme yapıldı.  
  
4. Hata Vurgusunda Eksik Karakter Sorunu:  
 - Noktalı virgül veya kapanan parantez eksik olduğunda metin sonu yerine bir önceki token üzerine vurgu yapıldı.  
  
5. “x = +6;” ve “x = 6;” Biçimlerinin Desteklenmesi:  
 - Parser atama kuralı 3 alt duruma bölündü (3A, 3B, 3C) ve bu biçimler desteklendi.

## 9. Sonuç ve Gelecek Geliştirmeler

Projede:  
- Lexer: 6 token tipi (KEYWORD, NUMBER, OPERATOR, PAREN, SEPARATOR, IDENTIFIER) başarıyla tespit edildi.  
- Parser: int, if, atama (x=x+6, x=+6, x=6) ve if bloğu (if(x>3){}) yapıları doğru tanındı.  
- errorStart/errorEnd ile hatalı token konumları yakalandı.  
- GUI: Real-time highlighting, hata altı çizili vurgulama, başlık güncelleme mekanizması tamamlandı.  
  
Gelecek Geliştirmeler:  
1. Yeni Token Tipleri: else, while, for, return vb. eklenebilir.  
2. Blok İçeriği: if ve döngü bloklarının içi dolu ifadeler desteklenebilir.  
3. Else-If Zinciri: if-else-if-else yapıları eklenebilir.  
4. Döngüler: while ve for döngüleri parse edilebilir.  
5. Yorum Satırları ve Dize (String) Desteği: // ve /\* \*/ yorumları ile string literal’ler işlenebilir.  
6. Semantic Analiz ve Sembol Tablosu: Tanımlı değişken kontrolü, tip denetimi eklenebilir.  
7. Kod Tamamlama ve Satır Numaraları: Autocomplete ve satır numarası desteği eklenebilir.  
8. GUI İyileştirmeleri: Tema seçimi, yazı tipi ve renk şeması ayarları, Dosya Aç/Kaydet menüleri.

## 10. Kaynakça ve Ekler

- Java Platform SE 11 - Swing Kütüphanesi Dokümantasyonu, Oracle  
- Java Regex Kullanımı, Oracle  
- “Lexer and Parser Implementation in Java” Makalesi  
- Programlama Dilleri Projesi PDF (Ders Koordinatörü)  
- IntelliJ IDEA Community Edition Documentation