# گزارش پروژه مترجم جاوا


استاد راهنما:

جناب دکتر پارسا

تهیه کننده:

عطیه سروی

95400053

در این پروژه ورودی ما می تواند کد به زبان فارسی یا انگلیسی باشد و به زبان دیگر ترجمه شود.

این کد به زبان جاوا در برنامه IntelliJ IDEA نوشته شده است.

با ران کردن این کد یک صفحه textArea باز می شود که در آن می توانیم کد Java را به زبان فارسی یا انگلیسی تایپ کنیم (با رعایت قوانین مربوط به زبان java ) سپس با فشردن یکی از button های موجود روی صفحه متن فارسی را به انگلیسی و یا برعکس تبدیل کرده و به عنوان خروجی بگیریم.

طرز عمل برنامه:

این برنامه از 3 کلاس Translatorو Main و Gui تشکیل شده که هر یک کاربرد خاصی دارد.

کلاس translator در واقع یک مترجم آنلاین است که با دو متد ترجمه کلمه را به از زبان انگلیسی به فارسی و برعکس انجام می دهد و کلمه ترجمه شده را به خروجی می دهد.

کلاس Gui محتوای گرافیکی برنامه را مهیا می‌کند . پس از تشکیل صفحه مورد نظر کد وارد شده در صفحه را می گیرد و آن را به صورت یک رشته به متد lexer در کلاس Main می دهد. حال به نحوه عملکرد این کلاس می پردازیم.

اولا که در این کلاس feature ی از کلاس Translator داریم برای ترجمه کلمات.

در این کلاس متد هایی داریم که هر یک عملی مخصوص را انجام می دهد.

مثلا متد isNumber(char ch) که یک کاراکتر را به عنوان ورودی می‌گیرد و اگر این کاراکتر شماره بود true برمی‌گرداند.

متد isCharSet در واقع مشخص می‌کند که آیا کاراکتر ورودی از عناصر operatorیا grouping هستند یعنی:

```
+     -     *     /     %     ^     &     |     !     ~
=     +=    -=    *=    /=    %=     ^=     ++     --
==    !=     <    >     &=   >>=    <<=    >=    <=
||    &&     >>     <<     ?:
```

و

```
[ ]    ( )    { }    ,    ;    .    "
```

متد isWord هم بررسی می‌کند که کاراکتر از حروف زبان انگلیسی است یا خیر.

متد isWordFa هم بررسی می‌کند که کاراکتر از حروف فارسی است یا خیر.

متد checkWord_engToFa هم ابتدا چک می‌کند که string ورودی جزوKeyWord های زبان Java نباشد اگر بود ترجمه مد نظر ما را برمی‌گرداند در غیر این صورت به کلاسtranslator می‌رود و لغت را ترجمه می‌کند و برمی‌گرداند.

checkWord_FaToEng هم همین کار را برای ورودی های فارسی انجام می‌دهد.

جدول(1) :کلمات کلیدی جاوا به فارسی و انگلیسی

| کلمه کلیدی انگلیسی | کلمه کلیدی به فارسی |
|---|---|
| Abstract | انتزاعی |
| Assert | تاکید |
| Boolean | بولی |
| Break | بشکن |
| Byte | بایت |
| Case | مورد |
| Catch | بگیر |
| Char | کاراکتر |
| Class | کلاس |
| Const | ثابت |
| Continue | ادامه_بده |
| Default | پیشفرض |
| Do | انجام_بده |
| Double | اعشاری_بلند |
| Else | وگرنه |
| Enum | قابل_شمارش |
| Extends | گسترش_میدهد |
| Final | نهایی |
| Finally | بالاخره |
| Float | اعشاری |
| For | برای |
| Goto | برو_به |
| If | اگر |
| Implements | پیاده_میکند |
| Import | وارد_کن |
| Instanceof | نمونه_از |
| Int | صحیح |
| Interface | اتصال |
| Native | محلی |
| New | جدید |
| Package | بسته |

| | |
|---|---|
| خصوصی | Private |
| حفاظتی | Protected |
| عمومی | Public |
| برگردان | Return |
| کوتاه | Short |
| ایستا | Static |
| مافوق | Super |
| جابه_جاکن | Switch |
| همزمان | Synchronized |
| همین | This |
| بینداز | Throw |
| می_اندازد | Throws |
| گذرا | Transient |
| امتحان_کن | Try |
| خالی | Void |
| فرار | Volatile |
| وقتی | While |
| چاپ_کن | println |
| المان_ها | args |
| بده_بیرون | out |
| سیستم | System |
| Io | Io |
| اسکن_کننده | Scanner |
| رشته | String |

نکته: فاصله بین کلمات فارسی با space پر شده که بتوان بین کلمات کلیدی و غیره فرق بگذاریم.

**نکته**: اگر کلمه ای در DICTIONARY وجود نداشته باشد خود تابع یک ترجمه خاص انجام می‌دهد مثلا ASASAرا "اسااس" ترجمه می‌کند.

نکته: اگر بخواهیم کلمه فارسی را به فارسی ترجمه کنیم و برعکس خروجی خود کلمه میشود.

در نهایت هم تابع lexer ورودی اش رشته‌ای است که از ما درtextArea وارد کردیم و سپس دکمه translate را فشرده‌ایم.این تابع این رشته را گرفته و کاراکتر به کاراکتر بررسی می‌کند که این رشته ها چه چیزی هستند یعنی کاراکتر ها را تا زمانی که space یا tab یا enter نباشد و همچنین digit و operatorو grouping نباشند به یک رشته خالی اضافه می‌کند و وقتی به این ها رسید آن رشته را

ترجمه کرده و ترجمه اش را به رشته نهایی اضافه می‌کند و رشته فعلی را خالی می‌کند وبرای کاراکتر های بعدی تا وقتی رشته ورودی تمام شود این کار را انجام میدهد.

نکته:به جای operator ها و grouping characters و enterو spaceو tabخودشان را قرار می‌دهد.

این تابع برای ترجمه از فارسی به انگلیسی و برعکس نوشته شده و بسته به این که کسی که ورودی را وارد می‌کند کدام دکمه translate را فشار دهد تابع درست فراخوانی می‌شود.

**قوانین لغوی جاوا به‌طور خلاصه:**

// : کامنت یک خطی

/* commemt*/ :کامنت چند خطی

/**doc*/ :کامنت برای مستند

Space: برای ایجاد فاصله بین token ها و افزایش خوانایی

هر رشته ای که لغت کلیدی نباشد با رعایت قواعد زیر شناسه یا ID است:

* نمی‌تواند با عدد شروع شود

*نمیتوند با operator شروع شود

*نباید وسطش space باشد

:Operators

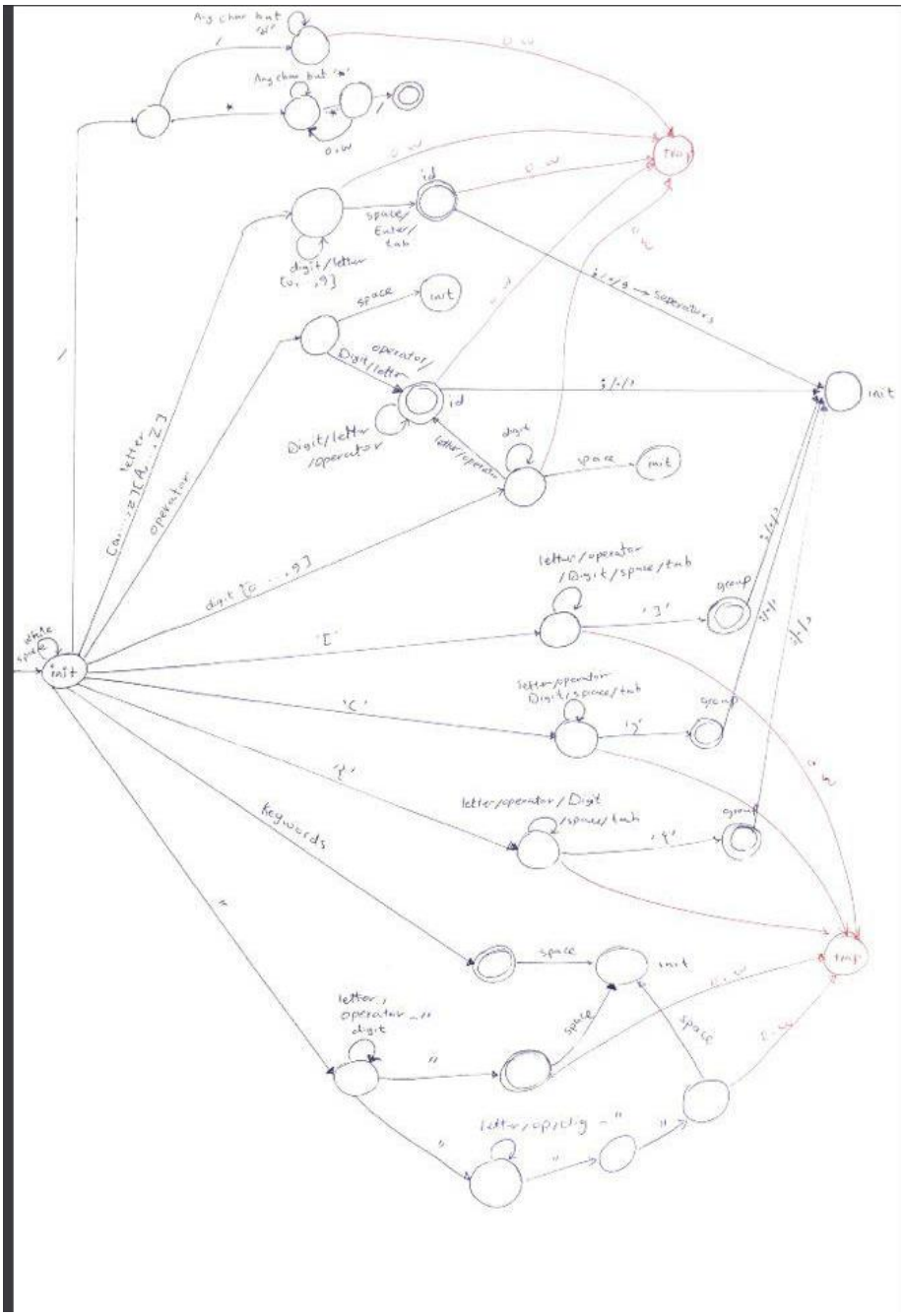
```
+     -     *     /     %     ^     &     |     !     ~
=     +=    -=    *=    /=    %=     ^=     ++     --
==    !=     <    >     &=   >>=    <<=    >=    <=
||    &&     >>     <<     ?:
```

:Separators

```
[ ]    ( )    { }    ,    ;    .    "
```

رشته ها که یا با یک " شروع و با یک " تمام می‌شود و یا با سه """ شروع و تمام می‌شود.

DFA قوانين لغوى جاوا:


Any char but '*'
Any char but '*'
0 or M
temp
id
space/ Enter/ tab
digit/letter [0, ,9]
init
space
operator/ Digit/letter
id
;/:/)
Digit/letter /operator
letter/operator
digit
space
init
letter/operator /Digit/space/tab
group
'I'
'C'
letter/operator Digit/space/tab
group
'f'
letter/operator/Digit /space/tab
group
keywords
space
init
temp
letter, operator _ digit
space
space
letter/op/dig - "
"
white space
init
letter [A...Z]
operator
digit [0 ...9]
;/:/)

Java lexer analyser code:

```
package
lexer;
		import java.util.ArrayList;
		import java.util.List;
		import java.util.Map;
		import java.util.TreeMap;
		import java.util.regex.Matcher;
		import java.util.regex.Pattern;
		import exceptions.AnalyzerException;
		import token.Token;
		import token.TokenType;
		public class Lexer {
			/** Mapping from type of token to its regular expression */
			private Map<TokenType, String> regEx;
			/** List of tokens as they appear in the input source */
			private List<Token> result;
			/**
			 * Initializes a newly created {@code Lexer} object
			 */
			public Lexer() {
				regEx = new TreeMap<TokenType, String>();
				launchRegEx();
				result = new ArrayList<Token>();
			}
			public void tokenize(String source) throws AnalyzerException {
				int position = 0;
				Token token = null;
				do {
					token = separateToken(source, position);
					if (token != null) {
						position = token.getEnd();
						result.add(token);
					}
				} while (token != null && position != source.length());
				if (position != source.length()) {
					throw new AnalyzerException("Lexical error at position # "+
		position, position);
				}
			}
			public List<Token> getTokens() {
				return result;
```

```
	}
	public List<Token> getFilteredTokens() {
		List<Token> filteredResult = new ArrayList<Token>();
		for (Token t : this.result) {
			if (!t.getTokenType().isAuxiliary()) {
				filteredResult.add(t);
			}
		}
		return filteredResult;
	}
	private Token separateToken(String source, int fromIndex) {
		if (fromIndex < 0 || fromIndex >= source.length()) {
			throw new IllegalArgumentException("Illegal index in the input
stream!");
		}
		for (TokenType tokenType : TokenType.values()) {
			Pattern p = Pattern.compile(".{" + fromIndex + "}" +
regEx.get(tokenType),
					Pattern.DOTALL);
			Matcher m = p.matcher(source);
			if (m.matches()) {
				String lexema = m.group(1);
				return new Token(fromIndex, fromIndex +
lexema.length(), lexema, tokenType);
			}
		}
		return null;
	}
	private void launchRegEx() {
		regEx.put(TokenType.BlockComment, "(/\\*.*?\\*/).*");
		regEx.put(TokenType.LineComment, "(//(.*?)[\r$]?\n).*");
		regEx.put(TokenType.WhiteSpace, "( ).*");
		regEx.put(TokenType.OpenBrace, "(\\().*");
		regEx.put(TokenType.CloseBrace, "(\\)).*");
		regEx.put(TokenType.Semicolon, "(;).*");
		regEx.put(TokenType.Comma, "(,).*");
		regEx.put(TokenType.OpeningCurlyBrace, "(\\{).*");
		regEx.put(TokenType.ClosingCurlyBrace, "(\\}).*");
		regEx.put(TokenType.DoubleConstant,
"\\b(\\d{1,9}\\.\\d{1,32})\\b.*");
		regEx.put(TokenType.IntConstant, "\\b(\\d{1,9})\\b.*");
		regEx.put(TokenType.Void, "\\b(void)\\b.*");
		regEx.put(TokenType.Int, "\\b(int)\\b.*");
```

```
                regEx.put(TokenType.Double, "\\b(int|double)\\b.*");
                regEx.put(TokenType.Tab, "(\\t).*");
                regEx.put(TokenType.NewLine, "(\\n).*");
                regEx.put(TokenType.Public, "\\b(public)\\b.*");
                regEx.put(TokenType.Private, "\\b(private)\\b.*");
                regEx.put(TokenType.False, "\\b(false)\\b.*");
                regEx.put(TokenType.True, "\\b(true)\\b.*");
                regEx.put(TokenType.Null, "\\b(null)\\b.*");
                regEx.put(TokenType.Return, "\\b(return)\\b.*");
                regEx.put(TokenType.New, "\\b(new)\\b.*");
                regEx.put(TokenType.Class, "\\b(class)\\b.*");
                regEx.put(TokenType.If, "\\b(if)\\b.*");
                regEx.put(TokenType.Else, "\\b(else)\\b.*");
                regEx.put(TokenType.While, "\\b(while)\\b.*");
                regEx.put(TokenType.Static, "\\b(static)\\b.*");
                regEx.put(TokenType.Point, "(\\.).*");
                regEx.put(TokenType.Plus, "(\\+{1}).*");
                regEx.put(TokenType.Minus, "(\\-{1}).*");
                regEx.put(TokenType.Multiply, "(\\*).*");
                regEx.put(TokenType.Divide, "(/).*");
                regEx.put(TokenType.EqualEqual, "(==).*");
                regEx.put(TokenType.Equal, "(=).*");
                regEx.put(TokenType.ExclameEqual, "(\\!=).*");
                regEx.put(TokenType.Greater, "(>).*");
                regEx.put(TokenType.Less, "(<).*");
                regEx.put(TokenType.Identifier, "\\b([a-zA-Z]{1}[0-9a-zA-
Z_]{0,31})\\b.*");
        }
}
```

مثال های ورودی به فارسی:

/* سلام دنیا.جاوا

/*

/* سلام دنیا.جاوا

```
/*

عمومی کلاس سلام دنیا

}

عمومی ایستا خالی اصلی(رشته[] المان_ها) {

سیستم.بده_بیرون.چاپ_کن("سلام جهان!");

{

{
```

بعد از ترجمه:

```
Hello The world.Java */

/*

public class Hello The world

}

} (public static void the original(String[] args

;("!System.out.println("Hello the world

{

{
```

یا:

```
عمومی کلاس فاکتوریل

}

عمومی ایستا خالی اصلی(رشته[] المان_ها)

{  نهایی صحیح NUM_فاکتورها = 100;

برای(صحیح من = 0; من < NUM_فاکتورها; من++)

سیستم.بده_بیرون.چاپ_کن( من + "! است " + فاکتوریل(من));

{


عمومی ایستا صحیح فاکتوریل(صحیح n)

{  صحیح نتیجه = 1;

برای(صحیح من = 2; من <= n; من++)

نتیجه *= من;

برگردان نتیجه;

{

{

ترجمه:

public class Factorial

}

(public static void the original(String[] args

{  ;final int NUM_ invoices = 100

(++for(int I = 0; I < NUM_ invoices; I

;((System.out.println( I + "! Is " + Factorial(I

{
```

```
(public static int Factorial(int n
    ;int result = 1  }
    (++for(int I = 2; I <= n; I
        ;result *= I
    ;return result
    {
{
```

**منابع:**


**Zetcode.com**

**Oracle.com**

**Github.com**

**how to program Java /Deitel كتاب**