

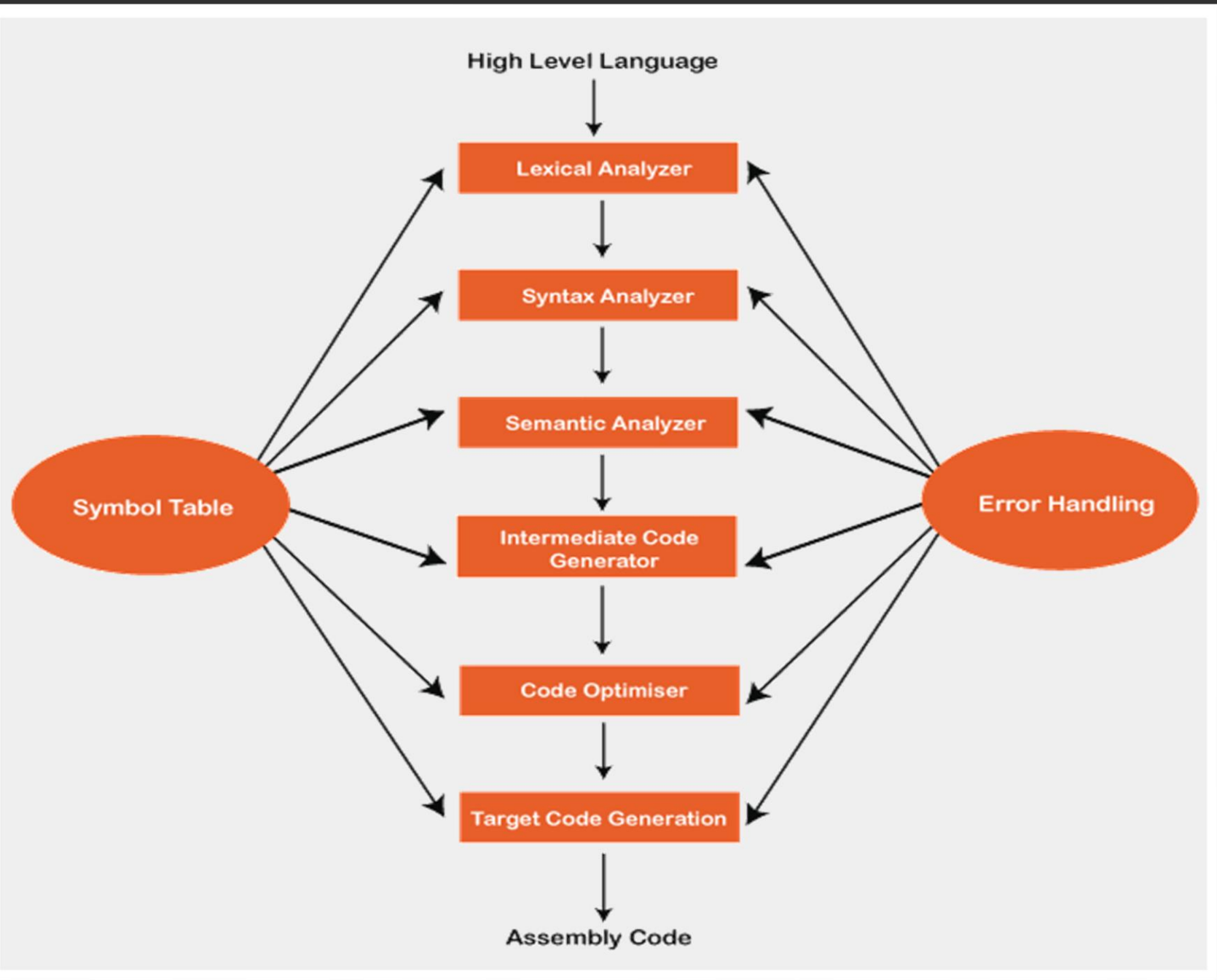
Symbol Table in Compiler

Compiler – 14022

Dr. Saeed parsa

What is symbol table?

- A crucial data structure in compiler
- Stores **information** about **identifiers**
- **Identifier** can be functions, classes, objects and variables
- **information** like name, type, scope, and memory location



Purpose of Symbol Table

- It is used by compiler to easily generate target code
- Compiler can achieve compile-time efficiency Symbol Table
- Used for type checking and scope resolution

How different parts of the compiler work with the symbol table?

1. Lexical Analysis

Creates Entries about
tokens of code

2. Syntax Analysis

Adds information regarding
attribute type, scope, etc

3. Semantic Analysis

Uses table to check for
semantics, like type
checking

4. Intermediate Code generation

Refers symbol table for
adding temporary variable
information

5. Code Optimization

Uses symbol table
information for optimization

6. Target Code generation

Generates code by using
address information of
identifier present in the
table

General Symbol Table Structure

ID	Name	Type	Value	Scope	...
1	X1	Int	5	Main	...
2	Y2	Float	6.2	Func1	...
...

Symbol Table Operations

- **Insertion:** When a new identifier is declared, it is added to the symbol table with its attributes.
- **Lookup:** The compiler searches the symbol table to retrieve an identifier's attributes.
- **Modification:** Attributes of an identifier can be updated, such as changing the scope level.
- **Deletion:** Identifiers that go out of scope are removed from the symbol table.

Example:

```
public class Calculator {  
    private int result = 0;  
  
    public void add(int number) {  
        result += number;  
    }  
  
    public int getResult() {  
        return result;  
    }  
  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
        calc.add(5);  
        System.out.println(calc.getResult());  
    }  
}
```

Name	Type	Scope	Address	Attributes
Calculator	Class	Global	0X5232	-
result	int	Calculator	0X3D32	private, initialized
add	Method	Calculator	0X3AA2	public
number	int	add	0xB968	parameter
getResult	Method	Calculator	0X299C	public
calc	Calculator	main	0XD778	local

Extended Symbol Table Builder (ESTBuilder)

EST_Hello.entity: 21 rows total

» Next

◀ Show all

▼ Sorting

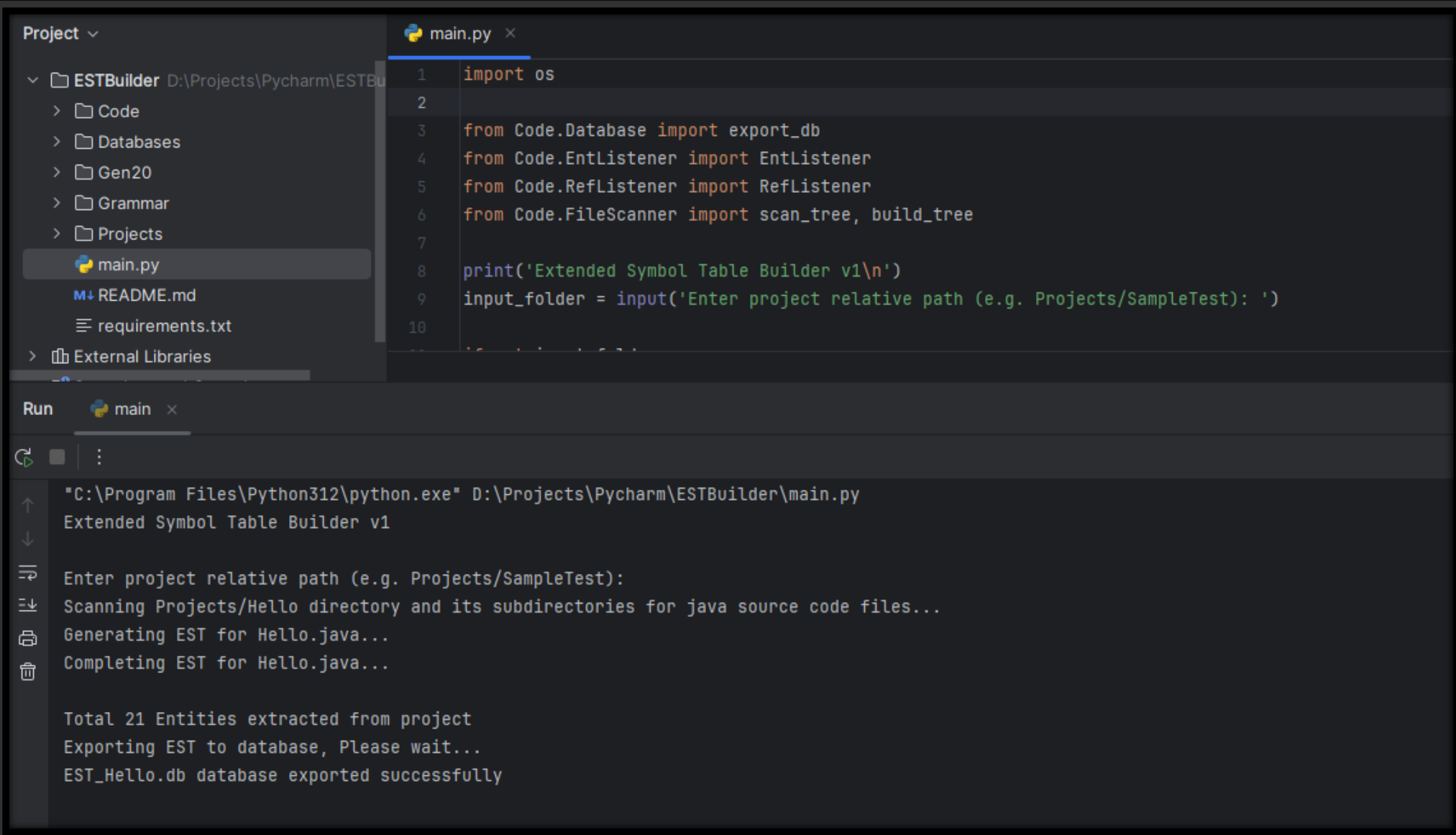
▼ Columns (10/10) ▼ Filter

id	uid	kind	name	type	value	scope	modifier	return_ty
1	a39b39cf8d6e4e57ab32f201aecd41b	file	Hello.java			jcg.zheng.demo.modifier		
2	e4de65a3ff094eaaab7048b664f90084	package	jcg.zheng.demo.modifier			Hello		
3	85acad02fcb84156825e07f54c403a81	import	java.util			jcg.zheng.demo.modifier		
4	93098a9fbe2246a5b9f318dd848c28b5	import	java.math			jcg.zheng.demo.modifier		
5	53dd26445333477394dd91ddc456a471	field	g	int	9	MyDirty	public static	
6	cd933040ad824f149e412eb6a03190cd	parameter	args	String		main		
7	b3d87e931b30433fbe17058674ea252d	block				main		
8	045be0c114c64e7ca98424f4f3d1d8be	variable	x	int	10	b3d87e93-1b30-433f-be17-058674ea252d		
9	4a1a630ff9fc46359ae2f9ac082bf5e8	variable	y	int	20	b3d87e93-1b30-433f-be17-058674ea252d	final	
10	ad51934933c442f28dfa66905672969e	method	main			MyDirty	public static	void
11	71a6606387394f49b754a3bfe7c99dfc	field	ij	int	9	InnerDirty		
12	dac63a3562f147b4850480586cc25906	class	InnerDirty	InnerDirty		MyDirty		
13	3ab2db4f4eb34cf0b9c5e73ce21e693b	parameter	a	int		add		
14	81b4c716af9948d6af0522cd146c12b4	parameter	b	int		add		
15	2b8168842c604a7eb51b541a1bbdbe9b	block				add		
16	1267899354e746b9bb14458a03ae225d	method	add			MyDirty	public static	int
17	cf17cb905cca4a9b9b37e3db545ad795	class	MyDirty	MyDirty		jcg.zheng.demo.modifier	public	
18	4ab9fcbe0cf84c3b9d951b0d0122fb3a	class	InnerInInterface	InnerInInterface		MyDirtyInterface		
19	d53ed87fc83c40f3b1516b28d7971491	interface	MyDirtyInterface	MyDirtyInterface		Hello.java	public	
20	47312c2343554697ac05bbe9fcbad06d	enum	Color	Color		Hello.java		
21	fecde86bc68842a9813740532a89273f	record	Point	Point		Hello.java		

<

>

Extended Symbol Table Builder (ESTBuilder)



The screenshot displays the PyCharm IDE interface. On the left, the 'Project' view shows the directory structure of the 'ESTBuilder' project, including folders like 'Code', 'Databases', 'Gen20', 'Grammar', and 'Projects', along with files 'main.py', 'README.md', and 'requirements.txt'. The 'main.py' file is selected and open in the editor. The code in 'main.py' includes imports for 'os', 'Code.Database', 'Code.EntListener', 'Code.RefListener', and 'Code.FileScanner', followed by a print statement and an input prompt for a project relative path. Below the editor, the 'Run' window shows the execution output for 'main.py'. The output indicates that the program is scanning the 'Projects/Hello' directory for Java source code files, generating an EST for 'Hello.java', and successfully exporting the EST to a database named 'EST_Hello.db'.

```
1 import os
2
3 from Code.Database import export_db
4 from Code.EntListener import EntListener
5 from Code.RefListener import RefListener
6 from Code.FileScanner import scan_tree, build_tree
7
8 print('Extended Symbol Table Builder v1\n')
9 input_folder = input('Enter project relative path (e.g. Projects/SampleTest): ')
10
```

Run main

"C:\Program Files\Python312\python.exe" D:\Projects\Pycharm\ESTBuilder\main.py
Extended Symbol Table Builder v1
Enter project relative path (e.g. Projects/SampleTest):
Scanning Projects/Hello directory and its subdirectories for java source code files...
Generating EST for Hello.java...
Completing EST for Hello.java...

Total 21 Entities extracted from project
Exporting EST to database, Please wait...
EST_Hello.db database exported successfully