***COURSE NAME:***

***COMPILER CONSTRUCTION***

***COURSE CODE:***

***CSSE 501***

***COURSE INCHARGE:***

***MS MARYAM FEROZE***

***GROUP MEMBERS:***

***ANNAS HAMEED QAZI(B22110106017)***

***XAIN MUHAMMAD KHAN(B22110106091)***

***SAMEER AHMED(B22110106069)***

* ***Int***
* ***Float***
* ***Double***
* ***Char***
* ***Boolean***
* ***Void***
* ***Const***
* ***Static***
* ***Return***
* ***If***
* ***Else***
* ***Switch***
* ***Case***
* ***Default***
* ***Break***
* ***Continue***
* ***For***
* ***While***
* ***Class***
* ***Public***
* ***Private***
* ***Protected***
* ***Interface***
* ***Extends***
* ***Implements***
* ***This***
* ***Super***
* ***New***
* ***Abstract***
* ***Override***
* ***Final***
* ***Constructor***
* ***Try***
* ***Catch***
* ***Finally***
* ***Throw***
* ***Throws***
* ***Final***
* ***Lambda***
* ***Package***
* ***Import***
* ***Main***
* ***Null***
* ***Var***
* ***Exports***

***New Unique keywords***

***1. "fuse" – Merge Two Objects into One***

***Functionality: Merges properties and methods of two objects into a single entity.***

***Example:***

***fuse obj1, obj2 into obj3;***

***// obj3 now contains all properties and methods of obj1 and obj2***

***2. "defer" – Postponed Execution***

***Functionality: Schedules a function or operation to execute after the current scope completes,***

***similar to JavaScript’s setTimeout.***

***Example:***

***defer (cleanupResources, 5000);***

***// cleanupResources will execute after 5 seconds***

***3. "recall" – Auto-Revert to Previous Value***

***Functionality: Stores the last value of a variable before reassignment, allowing it to be restored***

***when needed.***

***Example***

***int x = 5;***

***x = 10;***

***recall x; // x is now back to 5***

***Punctuators in Java***

1. **()**
2. **{}**
3. **[]**
4. **;**
5. **,**
6. **.**
7. **:**
8. **::**
9. **...**
10. **@**

***Operators in Java***

***Arithmetic Operators***

1. **+**
2. **-**
3. **\***
4. **/**
5. **%**
6. **++(increment)**
7. **-- (decrement)**

***Relational Operators***

1. **==**
2. **!=**
3. **>**
4. **<**
5. **>=**
6. **<=**

***Logical Operators***

1. **&&**
2. **||**
3. **!**

***Bitwise Operators***

1. **&**
2. **|**

***Arithmetic Operators***

1. **+=**
2. **-=**
3. **\*=**
4. **/=**
5. **%=**
6. **&=**
7. **|=**

***Relational Operators***

1. ***? :***

***Instanceof Operator***

1. ***instanceof***

***Lambda Operator***

1. ***->***

**Regular Expressions**

|  |
| --- |
| ***🔹 Basic Character Classes:*** |
| | **Regex** | **Description** | **Example Match** | | --- | --- | --- | | **.** | **Any character except newline** | **"a", "1", "@"** | | **\d** | **Any digit (0-9)** | **"5", "9"** | | **\D** | **Any non-digit** | **"a", "@", "Z"** | | **\w** | **Word character (a-z, A-Z, 0-9, \_)** | **"hello", "123", "test\_1"** | | **\W** | **Non-word character** | **"@", "#", " "** | | **\s** | **Whitespace (space, tab, newline)** | **" " (space), "\t" (tab)** | | **\S** | **Non-whitespace** | **"A", "9", "\_"** | |
| ***🔹 Anchors (Position Matching):*** |
| | **Regex** | **Description** | **Example Match** | | --- | --- | --- | | **^abc** | **Starts with "abc"** | **"abc123" (✅) / "123abc" (❌)** | | **xyz$** | **Ends with "xyz"** | **"testxyz" (✅) / "xyztest" (❌)** | | **\bword\b** | **Exact word match** | **" word " (✅) / "word123" (❌)** | | **\Babc\B** | **"abc" not at a word boundary** | **"1abc1" (✅) / " abc " (❌)** | |
| ***🔹 Quantifiers (Repetitions):*** |
| | **Regex** | **Description** | **Example Match** | | --- | --- | --- | | **a\*** | **"a" 0 or more times** | **"", "a", "aaaa"** | | **b+** | **"b" 1 or more times** | **"b", "bbbb" (✅) / "" (❌)** | | **c?** | **"c" 0 or 1 time** | **"", "c"** | | **d{3}** | **"d" exactly 3 times** | **"ddd" (✅) / "dd" (❌)** | | **e{2,4}** | **"e" 2 to 4 times** | **"ee", "eee", "eeee"** | |
|  |
|  |
| ***🔹 Escape Sequences (Special Characters):*** |
| | **Regex** | **Description** | **Example Match** | | --- | --- | --- | | **\n** | **Newline** | **"Hello\nWorld" → "Hello" (line 1), "World" (line 2)** | | **\t** | **Tab** | **"Hello\tWorld" → "Hello World" (tab space)** | | **\r** | **Carriage return** | **"Hello\rWorld" → World (overwrites "Hello")** | | **\f** | **Form feed** | **"Hello\fWorld" (used in printers, rarely in Java)** | | **\v** | **Vertical tab** | **"Hello\vWorld" (rarely used)** | | **\\** | **Escape backslash** | **"C:\\Users\\" → C:\Users\** | |

**JAVA IDENTIFIERS:**

**Java Identifier Rules**

* **A-Z, a-z, 0-9, \_ (underscore), $ (dollar) allowed**
* **With Digit (0-9) start not allowed**
* **Java reserved keywords not allowed (e.g., class, static, int)**
* **Whitespace (space, tab, newline) not allowed**
* **Special characters (@, #, !, etc.) not allowed**

**2️ Regular Expression for Java Identifiers**

Best regex to check java valid identifiers:

regex

^[a-zA-Z\_$][a-zA-Z\d\_$]\*$

📌 **Explanation:**

* ^ → Start of identifier
* [a-zA-Z\_$] → First character must be **letter, \_ (underscore) or $**
* [a-zA-Z\d\_$]\* → then after that letters, digits, \_, and $ are allowed
* $ → End of identifier

**3️ Java Code to Validate Identifiers**

import java.util.regex.\*;

public class Main {

public static boolean isValidIdentifier(String identifier) {

String regex = "^[a-zA-Z\_$][a-zA-Z\\d\_$]\*$";

return identifier.matches(regex);

}

public static void main(String[] args) {

String[] testIdentifiers = {"\_validID", "$money", "hello123", "1wrong", "int", "java@code"};

for (String id : testIdentifiers) {

System.out.println(id + " is valid? " + isValidIdentifier(id));

}

}

}

**Output:**

\_validID is valid? true

$money is valid? true

hello123 is valid? true

1wrong is valid? false (cant start with digit)

int is valid? true (Reserved keyword must be checked seperately)

java@code is valid? false (Special characters not allowed)

**4️ Reserved Keywords Check**

Make a set if you want to block java reserved keywords:

import java.util.\*;

public class Main {

public static boolean isReservedKeyword(String identifier) {

Set<String> keywords = new HashSet<>(Arrays.asList(

"abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class", "continue",

"default", "do", "double", "else", "enum", "extends", "final", "finally", "float", "for",

"if", "implements", "import", "instanceof", "int", "interface", "long", "native", "new",

"package", "private", "protected", "public", "return", "short", "static", "strictfp", "super",

"switch", "synchronized", "this", "throw", "throws", "transient", "try", "void", "volatile", "while"

));

return keywords.contains(identifier);

}

public static void main(String[] args) {

System.out.println(isReservedKeyword("int")); // true

System.out.println(isReservedKeyword("myVar")); // false

}

}

**5️ Implement in your language**

This logic can be put in lexer or parser

public static boolean isValidCustomLanguageIdentifier(String identifier) {

return isValidIdentifier(identifier) && !isReservedKeyword(identifier);

}

**Single -Line Comments:**

1. **#>**

**Multi-Line Comments:**

1. **/\*~ ~\*/**