2. A JavaCC parser that parses infix expressions with parentheses.

InfixExpressionParser Class

Package Declaration

package InfixExpression;

- Declares that this class belongs to the InfixExpression package. This organizes the code and avoids naming conflicts.

Imports

import java.util.Stack;

- Imports the Stack class, which is used for managing parentheses validation (push and pop operations).

Class Declaration

public class InfixExpressionParser {

- Declares the InfixExpressionParser class, which contains methods to validate infix expressions.

Validation Method

public static void validatation(String expression) throws Exception {

- A public static method that validates whether a given infix expression is correctly formed.
- Throws an Exception if the expression is invalid.

Local Variable Declaration

Stack<Character> parenthesesStack = new Stack<>();

boolean lastWasOperator = false;

boolean lastWasOperand = false;

- parenthesesStack: Tracks opening parentheses to ensure they match closing parentheses.
- lastWasOperator: Tracks if the last processed token was an operator.
- lastWasOperand: Tracks if the last processed token was an operand (number).

Iterate Through the Expression

for (int i = 0; i < expression.length(); i++) {

  char token = expression.charAt(i);

- Iterates over each character (token) in the input expression.

Handle Whitespace

```
if (Character.isWhitespace(token)) {
   continue;
}
```

- Skips spaces in the expression.

Handle Numbers

```
if (Character.isDigit(token)) {
   while (i + 1 < expression.length() && Character.isDigit(expression.charAt(i + 1))) {
      i++;
   }
   lastWasOperand = true;
   lastWasOperator = false;
}
```

- Checks if the character is a digit.
- Handles multi-digit numbers by iterating over consecutive digits.
- Sets flags: lastWasOperand to true and lastWasOperator to false.

Handle Opening Parentheses

```
} else if (token == '(') {
   parenthesesStack.push(token);
   lastWasOperator = false;
   lastWasOperand = false;
}
```

- Pushes (onto the stack.
- Resets flags since parentheses neither qualify as operators nor operands.

Handle Closing Parentheses

```
} else if (token == ')') {
   if (parenthesesStack.isEmpty()) {
```

```
        throw new Exception("Mismatched closing parenthesis.");
    }
    parenthesesStack.pop();
    lastWasOperator = false;
    lastWasOperand = true;
}
```

- Checks for a matching (in the stack.
- Throws an error if there’s no opening parenthesis.
- Pops the stack to match parentheses.

Handle Operators

```
} else if (isOperator(token)) {
    if (lastWasOperator || !lastWasOperand) {
        throw new Exception("Operator without operand.");
    }
    lastWasOperator = true;
    lastWasOperand = false;
}
```

- Validates operators: Ensures they follow an operand and aren’t consecutive.
- Throws an error if an operator appears without a preceding operand.

Handle Invalid Characters

```
} else {
    throw new Exception("Invalid character in expression: " + token);
}
```

- Throws an exception for unrecognized characters.

Final Checks

```
if (!parenthesesStack.isEmpty()) {
    throw new Exception("Mismatched opening parenthesis.");
}
```

```
if (lastWasOperator) {
    throw new Exception("Expression ends with an operator.");
}
```

- Ensures all opening parentheses are matched.
- Checks that the expression doesn’t end with an operator.

Helper Method

```
private static boolean isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}
```

- Determines if a character is one of the supported operators.

InfixExpressionParserTest Class

Package Declaration

```
package InfixExpression;
```

- Same package as the parser class.

Imports

```
import java.util.Scanner;
```

- Imports Scanner for user input.

Main Method

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
```

- The program entry point.
- Creates a Scanner object to read input from the user.

Input Loop

```
while (true) {
    System.out.println("\nEnter an infix expression:");
    String userExpression = scanner.nextLine().trim();
```

- Repeatedly prompts the user to input an infix expression.

- Trims whitespace from the input.

Validation and Feedback

```
try {
   InfixExpressionParser.validatation(userExpression);
   System.out.println("It is valid Expression");
} catch (Exception e) {
   System.out.println("It is Invalid Expression");
   System.out.println(" " + e.getMessage());
}
```

- Calls the validatation method to check the user's expression.
- Prints whether the expression is valid or invalid, along with an error message for invalid cases.

Resource Management

```
scanner.close();
```

- Ensures the Scanner resource is properly closed to avoid resource leaks.

Summary

- The InfixExpressionParser class validates infix expressions, ensuring correct use of parentheses, operators, and operands.
- The InfixExpressionParserTest class interacts with the user, prompting them to enter expressions and displaying validation results.