CDAC MUMBAI _JH
Lab Assignment

Section 1: Error-Driven Learning in Java

Objective: This assignment focuses on understanding and fixing common errors encountered inJava programming. By analyzing and correcting the provided code snippets, you will develop a deeper understanding of Java's syntax, data types, and control structures.

Instructions:

1. **Identify the Errors:** Review each code snippet to identify the errors or issues present.
2. **Explain the Error:** Write a brief explanation of the error and its cause.
3. **Fix the Error:** Modify the code to correct the errors. Ensure that the code compiles and runs asexpected.
4. **Submit Your Work**: Provide the corrected code along with explanations for each snippet.

Snippet 1:

```
public class Main {
    public void main(String[]
        args) {
        System.out.println("Hello,
        World!");
    }
}
```

Correct code:

```
public class Main {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

What error do you get when running this code?

Ans: Main method is not static in class Main, please define the main method as: public static void main(String[]args).

Snippet 2:

```
public class Main {
    static void main(String[]
        args) {
        System.out.println("Hello,
        World!");
    }
}
```

Correct code:
```
          public class Main {
               public static void main(String[] args){
                   System.out.println("Hello, World!");
          }
 }
```

What happens when you compile and run this code?
Ans: Here, public keyword is missing before static void main(String[] args) While running this code an error will be shown on the console that main method not found in class Main.

Snippet 3:
```
public class Main {
             public static int main(String[] args) {
                        System.out.println("Hello, World!");
                         return 0;
            }
}
```
Correct Code :
```
           public class Main {
                 public static void main(String[] args) {
                      System.out.println("Hello, World!");
                       return 0;

    }
```
What error do you encounter? Why is void used in the main method?
 main' method is not declared with a return type of 'void'
public static int main(String[] args) int replace with void

Why is void used in the main method?

The void keyword is used in the main method in Java to indicate that the method does not return any value. Here that's why void is specifically used in the main method.

Snippet 4:
```
 public class Main {
    public static void main() {
       System.out.println("Hello,
       World!");
    }
 }
```

Correct code:
```
 public class Main {
   public static void main(String[] args) {
             System.out.println("Hello, World!");
          }
 }
```

What happens when you compile and run this code? Why is String[] args needed?
error: can't find main(String[]) method in class: Main

Why is String[] args needed?
The String[] args parameter in the main method of a Java program allows the program to accept

command-line arguments. make program more flexible and customizable without needing to modify the code That’s why it's important.

Snippet 5:

```
public class Main {
   public static void main(String[] args) {
      System.out.println("Main method with
      String[] args");
   }
   public static void main(int[] args) {
      System.out.println("Overloaded main method with
      int[] args");
   }
}
```

Correct code:

```
public class Main {
        public static void main(String[] args) {
                System.out.println("Main method with String[] args");
                int[] intArgs = {1, 2, 3};
      main(intArgs);
        }

public static void main(int[] args) {
        System.out.println("Overloaded main method with int[] args");
        }
}
```

Can you have multiple main methods? What do you observe?
Yes,
Main method with String[] args
Overloaded main method with int[] args
What do you observe?
In the given provided code I got only got Main method with String[] args
As output. For a overloading main method we need create integer args in int[] intArgs = {1, 2, 3}; main(intArgs); in 1st main method. After compiling again got a Main method with String[] args, Overloaded main method with int[] args both as output

Snippet 6:

```
public class Main {
   public static void
      main(String[] args) {int x
      = y + 10;
      System.out.println(x);
   }
}
```

Correct code:

```
public class Main {
        public static void main(String[] args) {
                int y = 15;
                int x = y + 10;
                System.out.println(x);
   }
}
```

What error occurs? Why must variables be declared?
The variable y is used in the expression int x = y + 10; but it hasn't been declared or initialized anywhere in the code. In Java, all variables must be declared before they are used.
Why must variables be declared?
Type safety, memory allocation, Scope and Initialization.

Snippet 7:
```
public class Main {
   public static void
      main(String[] args) {int x
      = "Hello";
      System.out.println(x);
   }
}
```

Correct code:
```
public class Main {
        public static void main(String[] args) {
        String x = "Hello";
        System.out.println(x);
        }
}
```

What compilation error do you see? Why does Java enforce type safety?
String cannot be converted to int. integer cannot hold the string value.
Why does Java enforce type safety?
Java enforces type safety to prevent type-related errors at runtime by catching them during compilation. Java type safety imp because Preventing Errors at Compile Time, Code Clarity and Maintenance, memory management, Optimization by the Compiler.

Snippet 8:
```
public class Main {
   public static void
      main(String[] args) {
      System.out.println("Hello,
      World!"
   }
}
```

Correct code:
```
public class Main {
        public static void main(String[] args) {
           System.out.println("Hello, World!");

        }
}
```

What syntax errors are present? How do they affect compilation?
Missing Closing Parenthesis for System.out.println.
How do they affect compilation?
The compiler reads the code line by line and expects that all method calls are properly closed with matching parentheses. When it encounters the line System.out.println("Hello, World!", it doesn't find

the closing parenthesis), which causes the parser to become confused about where the method call ends.

Snippet 9:

```
public class Main {
   public static void
      main(String[] args) {int
      class = 10;
      System.out.println(class);
   }
}
Correct code:
public class Main {
         public static void main(String[] args) {
                    int myclass = 10;
                    System.out.println(myclass);
         }
}
```

What error occurs?
The errors occurs (not a statement, ';' expected, <identifier> expected, illegal start of expression) all stem from the fact that the keyword class is being used incorrectly.

Why can't reserved keywords be used as identifiers?
We can not use class as variable. In java we cannot use keywords as variable (class, if, for, while, etc) there are 52 keywords are reserved that we cannot use as varivable.
The errors occurs (not a statement, ';' expected, <identifier> expected, illegal start of expression) all stem from the fact that the keyword class is being used incorrectly.
The compiler gets confused and can't parse the line correctly because it sees class as part of the syntax for defining a class, not as a variable name.
class is a Reserved Keyword:
In Java, class is a reserved keyword used to define a class. Keywords in Java have special meanings and are part of the language's syntax. They cannot be used as variable names, method names, or any other identifier in your code.
Trying to use class as a variable name results in a syntax error because the compiler expects it to be used in its intended context (e.g., class MyClass { }).

Snippet 10:

```
public class
   Main { public
   void display()
   {
      System.out.println("No parameters");
   }
   public void display(int num) {
      System.out.println("With parameter:
      " + num);
   }
   public static void
      main(String[]args) {
       display();
      display(5);
   }
}
```

Correct code:
```
public class Main {
        public void display() {
                System.out.println("No parameters");
        }
        public void display(int num) {
                System.out.println("With parameter: " + num);
        }
        public static void main(String[] args) {
                Main m = new Main();
                m.display();
                m.display(5);
        }
}
```

What happens when you compile and run this code?
The main method is static, and it cannot directly call instance methods (display methods in this case) without an instance of the class.
Error message:
```
error: non-static method display() cannot be referenced from a static context
display();

error: non-static method display(int) cannot be referenced from a static context
display(5);
```

Is method overloading allowed?
Yes, Allowed in Java
method overloading is allowed in Java. Method overloading is when you have multiple methods in the same class with the same name but different parameter lists (different types or numbers of parameters).

Snippet 11:
```
public class Main {
   public static void
      main(String[] args) {int[]
      arr = {1, 2, 3};
      System.out.println(arr[5]);
   }
}
```

Correct code:
```
public class Main {
        public static void main(String[] args) {
        I       nt[] arr = {1, 2, 3};
                System.out.println(arr[2]);
        }
}
```
Error: just change the index value

What runtime exception do you encounter?
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3 at Main.main(Demo3.java:122)
```
Why does it occur?
Invalid Index**:** Arrays in Java are zero-indexed, meaning the index starts at 0 and goes up to length - 1.

If you try to access an index that is less than 0 or greater than or equal to the array's length, the JVM throws an ArrayIndexOutOfBoundsException to indicate that the index is invalid.
Bounds Checking**:** Java performs bounds checking on array accesses to ensure that you do not access memory outside of the array's allocated space. This is done to prevent potential runtime errors and security issues related to invalid memory access.

Snippet 12:

```
public class Main {
   public static void
      main(String[] args) {while
      (true) {
         System.out.println("Infinite Loop");
      }
   }
}
```

Correct code:

```
public class Main {
   public static void main(String[] args) {
      int a = 0;
      while (true) {
         System.out.println("Iteration: " + a);
         a++;
         if (a >= 5) {
            break; // Exit the loop when a reaches 5
         }
      }
   }
}
```

What happens when you run this code?

The while (true) loop will continuously execute because the condition true is always satisfied, meaning the loop never terminates.

Output: The code will continuously print "Infinite Loop" to the console without stopping. This will happen indefinitely, causing the program to run forever (or until it is manually terminated).

How can you avoid infinite loops?

Use proper loop conditions, increment counters, or include a break statement to ensure that the loop will eventually terminate.

Snippet 13:

```
public class Main {
   public static void
      main(String[] args) {String
      str = null;
      System.out.println(str.leng
      th());
   }
}
```

Correct code:

```
public class Main {
        public static void main(String[] args) {
                String str = null;
                if (str != null) {
                        System.out.println(str.length());
                }
                        else {
```

```
                    System.out.println("String is null");
                }
            }
}
```

What exception is thrown?

Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is null

Why does it occur?

The exception occurs because the code tries to call the length() method on a null reference, which does not point to any object.

Snippet 14:

```
public class Main {
   public static void
      main(String[] args) {
      double num = "Hello";
      System.out.println(num);
   }
}
```

Correct code:

```
public class Main {
        public static void main(String[] args) {
                double num = 10.5; // Assigning a valid double value
                System.out.println(num);
   }
}
```

What compilation error occurs?

String cannot be converted to double

```
        double num = "Hello";
```

Why does Java enforce data type constraints?

Java enforces data type constraints to prevent errors, improve code reliability, and ensure that operations on variables are type-appropriate and predictable.

Snippet 15:

```
public class Main {
        public static void main(String[] args) {
                int num1 = 10;
                double num2 = 5.5;
                int result = num1 + num2;
                System.out.println(result);
        }
}
```

Correct code:

What error occurs when compiling this code?

```
incompatible types: possible lossy conversion
int result = num1 + num2; System.out.println(result);
1 error
```

How should you handle different data types in operations?

using type casting we can handle different data type.

To handle different data types in operations typecasting needs to be done first in order to convert all data types in one form.

Snippet 16:

```
public class Main {
        public static void main(String[] args) {
                int num = 10;
                double result = num / 4;
                System.out.println(result);
        }
}
```

Correct code:

What is the result of this operation?

2.0

Is the output what you expected?

Yes, this is because the integer division truncates the decimal part, and then the result is assigned to a double variable.

Snippet 17:

```
 public class Main {
    public static void
      main(String[] args) {int a
      = 10;
      int b = 5;
      int result = a ** b;
      System.out.println(result);
    }
 }
```

Correct code:

```
public class Main {
        public static void main(String[] args) {
                int a = 10;
                int b = 5;
                double result= Math.pow(a, b);
                System.out.println(result);
        }
}
```

What compilation error occurs?

```
error: illegal start of expression
          int result = a ** b;
```

Why is the ** operator not valid in Java?

Java does not use the ** operator for exponentiation. Instead, Java provides a method in the Math class to perform exponentiation.

The designers of Java chose not to include an infix exponentiation operator like **, possibly to keep the set of operators simple and to avoid conflicts with existing operators.

Correct Way to Perform Exponentiation in Java:

To raise a number to the power of another in Java, you should use the Math.pow() method, which takes two arguments: the base and the exponent.

Snippet 18:

```
 public class Main {
    public static void
```

```
        main(String[] args) {int a
        = 10;
        int b = 5;
        int result = a + b * 2;
        System.out.println(result);
    }
}
```

Correct code:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        int result = (a + b) * 2; // Parentheses added to change precedence
        System.out.println(result); // Output will be 30
    }
}
```

What is the output of this code?

20

How does operator precedence affect the result?

The result is affected by operator precedence, where multiplication is performed before addition, leading to the final result. If the operations were performed strictly left to right without considering precedence, the result would have been different, but because of the correct precedence rules, the correct and expected output is 20.

Snippet 19:

```
public class Main {
    public static void
        main(String[] args) {int a
        = 10;
        int b = 0;
        int result = a / b;
        System.out.println(result);
    }
}
```

Correct code: try with these method

```
public class Main {
        public static void main(String[] args) {
                int a = 10;
                int b = 0;
                try {
                        int result = a / b;
                        System.out.println(result);
                }
                catch (ArithmeticException e) {
                        System.out.println("Division by zero is not allowed.");
                }
        }
}
```

What runtime exception is thrown?

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.main(Demo3.java:208)
```

Why does division by zero cause an issue in Java?

A divide by zero error generates a processor exception which triggers an interrupt. The interrupt is

"read" by the operating system and forwarded to the program if a handler is registered. Since Java registers a handler, it receives the error and then translates it into an ArithmeticException that travels up the stack.
Times

Snippet 20:

```
 public class Main {
          public static void main(String[] args) {
                    System.out.println("Hello, World")
          }
 }Correct code:
 public class Main {
          public static void main(String[] args) {
                    System.out.println("Hello, World");
          }
 }
```

What syntax error occurs?

```
error: ';' expected
            System.out.println("Hello, World")
```

Missing semicolon at end of line

How does the missing semicolon affect compilation?

Missing of Semi colon demarcates a statement—the compiler is confused by its absence, thinking multiple lines are in fact a single statement. This generates a compilation error, as almost always the combined multi-line expression is invalid

---

Snippet 21:

```
public class Main {
          public static void main(String[] args) {
                    System.out.println("Hello, World!");
// Missing closing brace here
}
```

Correct code:

```
public class Main {
          public static void main(String[] args) {
                    System.out.println("Hello, World!");
          }
}
```

What does the compiler say about mismatched braces?

Mismatched braces in Java typically result in a compilation error, as the Java compiler expects code blocks to be properly enclosed within matching braces ({}).

Snippet 22:

```
public class Main {
          public static void main(String[] args) {
                    static void displayMessage() {
                              System.out.println("Message");
                    }
          }
}
```

Correct code:

```
public class Main {
   public static void main(String[] args) {
      // Call the displayMessage method here
      displayMessage();
```

```
    }

    // Define the displayMessage method outside of main
    public static void displayMessage() {
        System.out.println("Message");
    }
}
```

What syntax error occurs?

```
error: illegal start of expression
            static void displayMessage() {
error: class, interface, enum, or record expected
}
2 errors
error: compilation failed
```

Can a method be declared inside another method?

In Java, a method cannot be declared inside another method. Java does not support nested methods or local methods. All methods must be defined at the class level, not inside other methods

---

Snippet 23:

```
public class Confusion {
        public static void main(String[] args) {
                int value = 2;
                switch(value) {
                case 1:
                        System.out.println("Value is 1");
                case 2:
                        System.out.println("Value is 2");
                case 3:
                        System.out.println("Value is 3");
                default:
                        System.out.println("Default case");
                }
        }
}
```

Error to Investigate: Why does the default case print after "Value is 2"? How can you prevent the program from executing the default case

Reason: The reason that the default case print after "Value is 2" is the missing of the break statement. To prevent the program from executing the default case we need to add break statement at the end of every case.

Correct code:

```
public class Confusion {
    public static void main(String[] args) {
        int value = 2;
        switch(value) {
            case 1:
                System.out.println("Value is 1");
                break; // Exit the switch block after this case
            case 2:
                System.out.println("Value is 2");
                break; // Exit the switch block after this case
            case 3:
                System.out.println("Value is 3");
                break; // Exit the switch block after this case
            default:
                System.out.println("Default case");
                break; // Exit the switch block after this case
        }
    }
}
```

Snippet 24:

```
public class MissingBreakCase {
        public static void main(String[] args) {
                int level = 1;
                switch(level) {
                        case 1:
                                System.out.println("Level 1");
                        case 2:
                                System.out.println("Level 2");
                        case 3:
                                System.out.println("Level 3");
                        default:
                                System.out.println("Unknown level");
                }
        }
}
```

Error to Investigate: When level is 1, why does it print "Level 1", "Level 2", "Level 3", and "Unknown level"? What is the role of the break statement in this situation?

ans

The reason that it prints" Level 1","Level 2","Level 3" and "Unknown level" is that due to missing of break statement.

The role of break statement in this situation is to terminate the code once the given condition is satisfied.

Snippet 25:

```
public class Switch {
        public static void main(String[] args) {
                double score = 85.0;
                        switch(score) {
                                case 100:
                                        System.out.println("Perfect score!");
                                        break;
                                case 85:
                                        System.out.println("Great job!");
                                        break;
                                default:
                                        System.out.println("Keep trying!");
                        }
                }
}
```

Error to Investigate:

Why does this code not compile?

The code does not compile because the switch statement in Java does not support double (or float) values.

What does the error tell you about the types allowed in switch expressions?

Ans: The switch statement can only be used with the following types: byte, short, int, char, enum, String and var.

How can you modify the code to make it work?

Correct code:

```
public class Switch {
	public static void main(String[] args) {
		double score = 85.0;
		int scoreInt = (int) score; // Cast the double to int
			switch(scoreInt) {
				case 100:
					System.out.println("Perfect score!");
					break;
				case 85:
					System.out.println("Great job!");
					break;
				default:
					System.out.println("Keep trying!");
			}
	}
}
```

Snippet 26:

```
public class Switch {
	public static void main(String[] args) {
		int number = 5;
		switch(number) {
			case 5:
				System.out.println("Number is 5");
				break;
			case 5:
				System.out.println("This is another case 5");
				break;
			default:
				System.out.println("This is the default case");
		}
	}
}
```

Error to Investigate:

Why does the compiler complain about duplicate case labels?

The compiler complains about duplicate case labels because: In Java, within a switch statement, each case label must be unique within the switch block. The compiler complains about duplicate case labels because each case label must be distinct; otherwise, the compiler cannot determine which block of code to execute.

What happens when you have two identical case labels in the same switch block?

When you have two identical case labels in the same switch block in Java, the compiler will generate an error. This is because having duplicate case labels creates ambiguity in which block of code should be executed, making it impossible for the compiler to determine which code should run when the switch statement matches that value.

Correct code:

```
public class Switch {
	public static void main(String[] args) {
		int number = 5;
		switch(number) {
```

```
                case 5:
                        System.out.println("Number is 5");
                        // No need for another case 5
                        break;
                default:
                        System.out.println("This is the default case");
            }
    }
}
```

Section 2: Java Programming with Conditional Statements

Question 1: Grade Classification

Write a program to classify student grades based on the following criteria:

- If the score is greater than or equal to 90, print "A"
- If the score is between 80 and 89, print "B"
- If the score is between 70 and 79, print "C"
- If the score is between 60 and 69, print "D"
  If the score is less than 60, print "F

Program:

```
class Grade{
        public static void main(String args[]){
                int marks = 50;
                if(marks>=90){
                        System.out.println("Grade A");
                }
                else if(marks>=80 && marks<=89){
                        System.out.println("Grade B");
                }
                else if(marks>=70 && marks<=79){
                        System.out.println("Grade C");
                }
                else if(marks>=60 && marks<=69){
                        System.out.println("Grade D");
                }
                else if(marks<60){
                        System.out.println("Grade F");
                }
        }
}
```

Output:

```
        Grade F
```

Question 2: Days of the Week

Write a program that uses a nested switch statement to print out the day of the week based on an integer input (1 for Monday, 2 for Tuesday, etc.). Additionally, within each day, print whether itis a weekday or weekend.

Program:

```
class Days{
        public static void main(String args[]){
                int day = 4;
                switch(day){
                        case 1 :
                                System.out.println("Today is Monday");
                                break;
                        case 2 :
                                System.out.println("Today is Tuesday");
                                break;
                        case 3 :
                                System.out.println("Today is Wednesday");
                                break;
                        case 4 :
                                System.out.println("Today is Thursday");
                                break;
                        case 5 :
                                System.out.println("Today is Friday");
                                break;
                        case 6 :
                                System.out.println("Today is Saturday");
                                break;
                        case 7 :
                                System.out.println("Today is Sunday");
                        break;
                        case 8 :
                                System.out.println("Invalid Day");
                                break;
                }
        }
}
```

Output:

```
        Today is Thursday
```

Question 3: Calculator

Write a program that acts as a simple calculator. It should accept two numbers and an operator(+, -, *, /) as input. Use a switch statement to perform the appropriate operation. Use nested if- else to check if division by zero is attempted and display an error message.

Program:

```
 import java.util.Scanner;

class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double num1, num2, result;
        char operator;

        System.out.println("Enter first number: ");
        num1 = sc.nextDouble();

        System.out.println("Enter operator (+, -, *, /): ");
        operator = sc.next().charAt(0);

        System.out.println("Enter second number: ");
        num2 = sc.nextDouble();

        if (operator == '+') {
            result = num1 + num2;
        } else if (operator == '-') {
            result = num1 - num2;
        } else if (operator == '*') {
            result = num1 * num2;
        } else if (operator == '/') {
            if (num2 != 0) {
                result = num1 / num2;
            } else {
                System.out.println("Error: Division by zero is not allowed");
                return;
            }
        } else {
            System.out.println("Invalid operator");
            return;
        }

        System.out.println("Result: " + result);
    }
}
```

Output:

```
        Enter first number: 5
        Enter operator (+, -, *, /):  -
        Enter second number: 4
        Result: 1.0
```

Question 4: Discount Calculation

Write a program to calculate the discount based on the total purchase amount. Use the following criteria:

- If the total purchase is greater than or equal to Rs.1000, apply a 20% discount.
- If the total purchase is between Rs.500 and Rs.999, apply a 10% discount.
- If the total purchase is less than Rs.500, apply a 5% discount.

Program:

```
class Discount {
    public static void main(String[] args) {
        int price = 200;
        double total;
        boolean membership = false;

        if (price >= 1000) {
            if (membership) {
                total = price * 0.25;
            } else {
                total = price * 0.2;
            }
        } else if (price >= 500 && price <= 999) {
            if (membership) {
                total = price * 0.15;
            } else {
                total = price * 0.1;
            }
        } else { // price < 500
            if (membership) {
                total = price * 0.1;
            } else {
                total = price * 0.05;
            }
        }

        System.out.println("Total : " + total);
    }
}
```

Output:

```
        Total: 10.0
```

Question 5: Student Pass/Fail Status with Nested Switch

Write a program that determines whether a student passes or fails based on their grades in three subjects. If the student scores more than 40 in all subjects, they pass. If the student fails in one or more subjects, print the number of subjects they failed in.

Program:

```
public class StudentPassFail {
    public static void main(String[] args) {
        // Example grades for three subjects
        int grade1 = 45;  // Score for Subject 1
        int grade2 = 38;  // Score for Subject 2
        int grade3 = 50;  // Score for Subject 3

        // Initialize the count of failed subjects
        int failedCount = 0;

        // Array to store grades
        int[] grades = {grade1, grade2, grade3};

        // Loop through each subject
        for (int subject = 0; subject < grades.length; subject++) {
            int grade = grades[subject];

            // Nested switch case to check grade for each subject
            switch (subject) {
                case 0: // Subject 1
                case 1: // Subject 2
                case 2: // Subject 3
                    // Inner switch to determine if the grade is passing or failing
                    switch (grade) {
```

```
            default:
                // If grade is more than 40, it is a pass
                // No action needed here
                break;
            case 40:
            case 39:
            case 38:
            case 37:
            case 36:
            case 35:
            case 34:
            case 33:
            case 32:
            case 31:
            case 30:
            case 29:
            case 28:
            case 27:
            case 26:
            case 25:
            case 24:
            case 23:
            case 22:
            case 21:
            case 20:
            case 19:
            case 18:
            case 17:
            case 16:
            case 15:
            case 14:
            case 13:
            case 12:
            case 11:
            case 10:
            case 9:
            case 8:
            case 7:
            case 6:
            case 5:
            case 4:
            case 3:
            case 2:
            case 1:
            case 0:
                // If grade is 40 or less, consider as fail
                failedCount++;
                break;
        }
        break;
    default:
        // Should not occur
        System.out.println("Unexpected subject index.");
        break;
}
```

```
        }

        // Determine and print the result based on the count of failed subjects
        if (failedCount == 0) {
            System.out.println("The student passes.");
        } else {
            System.out.println("The student failed in " + failedCount + " subject(s).");
        }
    }
}
```