}

Assigment :2

Section 1: Error-Driven Learning in Java

Snippet 1:

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

 What error do you get when running this code?

Ans: The method signature for the main method in Java should be public static void main(String[] args). main method does not have the static keyword

Snippet 2:

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

 What happens when you compile and run this code?

Ans : code will compile successfully

Snippet 3:

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

 What error do you encounter? Why is void used in the main method?

Ans: The code will compile successfully because the syntax is correct

When you attempt to run this code, you will get an error message from the Java Runtime Environment (JRE), as it does not recognize the main method with an int return type as the entry point for the application.

The main method must be void because it is designed to be the starting point of a Java application and is not expected to return any value to the operating system. The method's purpose is to start the execution of the program, and its return value is not used by the Java Runtime Environment.

Snippet 4:

```
public class Main {

 public static void main() {

 System.out.println("Hello, World!");

 }

}
```

□ What happens when you compile and run this code? Why is String[] args needed?

Ans: The code will compile without errors because the syntax of the main method in this code is valid Java but it gives run time error bcz (JRE) will not recognize the main method because it does not match the required signature. to serve as the entry point for the application. The String[] args parameter allows the program to accept and handle command-line arguments.

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");

 }

}
```

□ Can you have multiple main methods? What do you observe?

Ans: multiple main methods are allowed in java but they must have different parameter lists. This is known as method overloading. Java allows you to define multiple methods with the same name as long as their parameter lists differ

Snippet 6:

```
public class Main {

 public static void main(String[] args) {

 int x = y + 10;

 System.out.println(x);

 }

}
```

 What error occurs? Why must variables be declared?

Ans: compilation error is occurred. The variable y is used in the expression int x = y + 10;, but it has not been declared or initialized anywhere in the code.

Why must variables be declared?

Java is a statically-typed language, which means that the type of every variable must be known at compile-time. Declaring variables ensures that the compiler knows what kind of data the variable will hold, enabling type checking and preventing errors.

Declaring a variable allows the Java runtime to allocate appropriate memory for the variable based on its type.

Explicitly declaring variables helps in understanding the code and maintaining it. It makes it clear what type of data is being handled and how it should be used.

Snippet 7:

```
public class Main {

 public static void main(String[] args) {

 int x = "Hello";

 System.out.println(x);

 }

}
```

 What compilation error do you see? Why does Java enforce type safety?

Ans: The error occurs because we are trying to assign a String value ("Hello") to a variable of type int. we cannot assign a value of one type (in this case, String) to a variable of another type (int) without an explicit conversion.

Type safety helps prevent type-related errors that could lead to runtime exceptions or unpredictable behavior. By ensuring that variables and expressions are used with the correct types, Java minimizes the chances of type mismatch errors.

Snippet 8:

```
public class Main {

 public static void main(String[] args) {

 System.out.println("Hello, World!"

 }

}
```

 What syntax errors are present? How do they affect compilation?

Ans: The code has a missing closing parenthesis and a missing semicolon in the System.out.println statement.

These syntax errors will prevent the Java compiler from successfully compiling the code. The compiler will produce error messages indicating that the syntax is incorrect and will not generate the bytecode necessary to run the program

Snippet 9:

```
public class Main {

 public static void main(String[] args) {

 int class = 10;

 System.out.println(class);

 }

}
```

 What error occurs? Why can't reserved keywords be used as identifiers?

Ans: **Error**: The code results in a compilation error because class is a reserved keyword in Java and cannot be used as an identifier for a variable.

Reserved Keywords: Reserved keywords have specific roles in the language's syntax and cannot be used as identifiers to avoid ambiguity, ensure consistency, and maintain code readability.

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);
 }
}
```

 What happens when you compile and run this code? Is method overloading allowed?

Ans: The code fails to compile because display() and display(int) are instance methods, and we are trying to call them from a static method (main) without an instance of the Main class.Method overloading is allowed in Java, and it works correctly when the method calls are made from appropriate contexts.

Snippet 11:

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

 What runtime exception do you encounter? Why does it occur?

Ans: ArrayIndexOutOfBoundsException occurs because the code attempts to access an array index (5) that is outside the valid range of indices for the array. This exception occurs due to bounds checking performed by Java

Snippet 12:

```
public class Main {
 public static void main(String[] args) {
 while (true) {
```

```
 System.out.println("Infinite Loop");

 }

 }

}
```

 What happens when you run this code? How can you avoid infinite loops?

Ans: this code will continuously print Infinite Loop. To avoid infinite loops, you can include a condition that will eventually terminate the loop, such as using a counter or a specific exit condition that evaluates to false at some point.

Snippet 13:

```
public class Main {

 public static void main(String[] args) {

 String str = null;

 System.out.println(str.length());

 }

}
```

 What exception is thrown? Why does it occur?

Ans: This code throws a NullPointerException because we're trying to call the length() method on a null reference. Since str is null, it doesn't point to an actual String object, leading to the exception when accessing its method.

Snippet 14:

```
public class Main {

 public static void main(String[] args) {

 double num = "Hello";

 System.out.println(num);

 }

}
```

 What compilation error occurs? Why does Java enforce data type constraints?

Ans: compilation error: incompatible types. we cannot assign a String value to a double variable because they are incompatible data types. Java enforces data type constraints to ensure type safety and prevent runtime errors, making sure operations are performed on compatible data types.

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);
 }
}
```

 What error occurs when compiling this code? How should you handle different data types in operations?

Ans: compilation error: incompatible types This occurs because we are trying to assign the result of an arithmetic operation involving a double to an int variable, which may lead to a loss of precision.

To handle different data types in operations properly:

1. **Use Type Casting:** Explicitly convert the double result to an int if you are sure that precision loss is acceptable: ex: int result = (int) (num1 + num2);
2. **Use a Compatible Data Type for Results:** Change the type of the result variable to double to accommodate the result of the arithmetic operation:ex: double result = num1 + num2;

Snippet 16:

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

 What is the result of this operation? Is the output what you expected?

Ans: The result of the operation will be 2.0. This happens because num / 4 performs integer division first, which results in 2, and then this integer value is implicitly converted to a double, so result is 2.0

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);

 }

}
```

 What compilation error occurs? Why is the ** operator not valid in Java?

Ans: The code will not compile due to a compilation error: not a statement because ** is not a valid operator in Java. Java does not support the ** operator for exponentiation; instead of that we can use Math.pow(a, b) for calculating powers.

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);

 }

}
```

 What is the output of this code? How does operator precedence affect the result?

Ans: The output of this code is 20 first the multiplication is perform then addition will be perform

So, the precedence of operators ensures that multiplication is done before addition, affecting the final result.

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);

 }

}
```

 What runtime exception is thrown? Why does division by zero cause an issue in Java?

Ans: The code throws an ArithmeticException with the message "divide by zero". Division by zero is not allowed in Java because it results in an undefined mathematical operation, leading to a runtime exception to indicate that the operation cannot be completed.

Snippet 20:

```
public class Main {

 public static void main(String[] args) {

 System.out.println("Hello, World")

 }

}
```

 What syntax error occurs? How does the missing semicolon affect compilation?

Ans: Missing Semicolon syntax error will be occurred. In Java, semicolons are used to terminate statements. Adding the semicolon ensures the code is correctly terminated and will compile and run as expected

Snippet 21:

```
public class Main {

 public static void main(String[] args) {

 System.out.println("Hello, World!");

 // Missing closing brace here
```

}

 What does the compiler say about mismatched braces?

Ans: When the code has a missing closing brace (}) in Java, the compiler will report an error indicating mismatched braces. Specifically, it will indicate that it reached the end of the file (EOF) or encountered an unexpected token because it expects a closing brace to match an earlier opening brace.

Snippet 22:

```
public class Main {

 public static void main(String[] args) {

 static void displayMessage() {

 System.out.println("Message");

 }

 }

}
```

 What syntax error occurs? Can a method be declared inside another method?

Ans: The compiler will produce an error indicating that method declarations are not allowed within 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?

Ans: In the provided code, the default case prints after "Value is 2" because the switch statement falls through each case until it encounters a break statement or reaches the end of the switch block. In Java, if a break statement is not used, execution continues through all subsequent cases, including the default case, regardless of which case was matched.

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: When level is 1, the code prints "Level 1", "Level 2", "Level 3", and "Unknown level" because of the lack of break statements in the switch block. In Java, if a break statement is not used at the end of a case block, the execution continues through all subsequent case blocks until a break is encountered or the switch block ends.

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? What does the error tell you about the

types allowed in switch expressions? How can you modify the code to make it work?

Ans: The code does not compile because Java's switch statement does not support double values as case labels. In Java, the switch statement can only be used with byte, short, int, char, enum types, and String objects (since Java 7). Floating-point types, such as double, are not supported for switch expressions.

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? What

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

Ans: The compiler complains about duplicate case labels because each `case` label in a `switch` block must be unique. Having two identical `case` labels (e.g., `case 5:`) within the same `switch` block is not allowed and results in a compilation error

Q.1

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"

```java
public class Grade {
    public static void main(String args[]) {
        int marks = 85;
        if (marks >= 90) {
            System.out.println("Grade A");
        } else if (marks >= 80) {
            System.out.println("Grade B");
        } else if (marks >= 70) {
            System.out.println("Grade C");
        } else if (marks >= 60) {
            System.out.println("Grade D");
        } else {
            System.out.println("Grade F");
        }
    }
}
```

Q.2

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 it is a weekday or weekend.

```java
public class Switchex {
    public static void main(String args[]) {
        int day = 7;
        switch (day) {
            case 1:
                System.out.println("monday");
                break;
            case 2:
                System.out.println("Teusday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("invalid day");
                break;
        }
    }
}
```

Q.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 ifelse to check if division by zero is attempted and display an error m

```
import java.util.Scanner;

public class calculater {
    public static void main(String[]args) {
        char operator;
        double num1, num2, result;

        Scanner input = new Scanner(System.in);
        System.out.println("Choose an operator +,-,*,/ : ");
        operator = input.next().charAt(0);
        System.out.println("Enter first number:");
        num1 = input.nextDouble();

        System.out.println("Enter Second number:");
        num2 = input.nextDouble();

        switch (operator) {
            case '+':
                result = num1 + num2;
                System.out.println("Result is:" + result);
                break;

            case '-':
                result = num1 - num2;
                System.out.println("Result is:" + result);
                break;

            case '*':
                result = num1 * num2;
                System.out.println("Result is:" + result);
                break;

            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                    System.out.println("Result is:" + result);
                } else {
                    System.out.println("Error! Division by zero is not
allowed");
```

```java
                }
                break;
            default:
                System.out.println("Invalid operator");
                break;
        }
    }
}
```

Q.4

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.

Additionally, if the user has a membership card, increase the discount by 5%.

```java
import java.util.Scanner;

public class Calculator2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter total amount:");
        double totalpurches = sc.nextDouble();

        System.out.println("do you have membership card (y/n): ");
        String membershipcard = sc.next().trim().toLowerCase();

        double discount = 0.0;
        if (totalpurches >= 1000) {
            discount = 20.0;
        } else if (totalpurches >= 500) {
            discount = 10.0;
        } else {
            discount = 5.0;
        }
        if (membershipcard.equals("y")) {
            discount = discount + 5.0;
        }
        double discountAmount = (discount / 100) * totalpurches;
        double finalAmount = totalpurches - discountAmount;

        System.out.printf("original purches Amount: %.2f%n", totalpurches);
        System.out.printf("Discount Applied: %.2f%%%n", discount);
        System.out.printf("Discount Amount:%.2f%n", discountAmount);
        System.out.printf("final Amount After Discount:%.2f%n", finalAmount);

        sc.close();
    }

}

Output:
enter total amount:
1000
do you have membership card (y/n):
y
original purches Amount: 1000.00
Discount Applied: 25.00%
Discount Amount:250.00
final Amount After Discount:750.00
```

Q.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.

```
import java.util.*;

public class StudentScore {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the grade for subject 1: ");
        int grade1 = sc.nextInt();

        System.out.print("Enter the grade for subject 2: ");
        int grade2 = sc.nextInt();

        System.out.print("Enter the grade for subject 3: ");
        int grade3 = sc.nextInt();

        int failsub = 0;

        if (grade1 <= 40) {
            failsub++;
        }
        if (grade2 <= 40) {
            failsub++;
        }
        if (grade3 <= 40) {
            failsub++;
        }

        if (failsub == 0) {
            System.out.println("the student passes. ");
        } else {
            System.out.printf("the student fails in %d subject(s).%n",
failsub);
        }

    }

}

Output1:

Enter the grade for subject 1: 40
Enter the grade for subject 2: 35
Enter the grade for subject 3: 87
the student fails in 2 subject(s).
```