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 : -

main method should be static

Correct code –

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

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: -

The code compiles successfully because the syntax is correct. The program will not run because the main method is missing the public modifier. The Java runtime requires the main method to be defined as public static void main(String[] args).

Correct code –

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

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

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 there is no syntax error in the code. But while executing the code the Java runtime will look for a method with the signature public static void main(String[] args), which has a void return type. Since the main method in the code returns an int instead of void, the runtime won't recognize it as the entry point. As soon as the main method terminates the java code terminates too. Hence it doesn't make any sense to return from the main method as JVM can't do anything with its return value.

Correct code :-

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

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 successfully because there is no syntax error. But while executing the code code the Java runtime will look for a method with the signature public static void main(String[] args). String args[] in Java is used to pass command-line arguments to a Java program.

Correct code : -

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

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 :-

The code will compile successfully because both methods are syntactically correct, and method overloading is allowed in Java. We have two main methods, each with a different parameter type (String[] and int[]). The JVM will always use the public static void main(String[] args) method as the entry point for program execution. This is because the JVM is designed to look for this specific method signature to start the application. The overloaded main method with int[] args will not be executed automatically. It can only be

called explicitly within the code by another method or from within the String[] args version of main.

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 : -

The error that occurred in this code is a compilation error due to the use of undeclared variable. We cannot use any variable without declaration and initialization.

Correct code :-

```
public class Main {

public static void main(String[] args) {

int y=2;

int x = y + 10;

System.out.println(x);

}

}
```

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:-

This code gives the incompatible type error. Java is a statically typed language, meaning every variable must be declared with a specific data type, and only values of that type can be assigned to it. So the datatype of variable x must be String. Java enforces type safety to prevent errors at runtime, ensure data integrity.

Correct code :-

```
public class Main {

public static void main(String[] args) {

String x = "Hello";

System.out.println(x);

}

}
```

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:-

It gives syntax error - Missing Closing Parenthesis) in System.out.println

Missing Semicolon ; at the End of the System.out.println Statement

Correct code :-

```
public class Main {

public static void main(String[] args) {

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

}

}
```

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 :-

The given Java code snippet contains a compilation error because it uses a reserved keyword as an identifier. In Java, class is a reserved keyword that is part of the language syntax. Keywords in Java have predefined meanings and are used to define the structure and syntax of the Java language. The class keyword is specifically used to declare a class, which is a fundamental building block in Java programming. Java does not allow keywords to be used as identifiers because it would cause ambiguity and confusion in the code.

Correct code :-

```
public class Main {

public static void main(String[] args) {

int num = 10;

System.out.println(class);

}

}
```

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 :-

It gives compile time error that is non-static methods cannot be referenced from a static context. So we need to declare these methods as a static method. Method overloading is allowed in Java. Method overloading is when multiple methods have the same name but different parameter lists.

Correct code :-

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

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 :-

The given Java code snippet will compile successfully but will throw a runtime exception when executed. When this code is executed, it will throw an ArrayIndexOutOfBoundsException. An ArrayIndexOutOfBoundsException occurs when you try to access an array index that is outside the range of valid indices for the array.

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 :-

When you run this code, it will enter an infinite loop. The while (true) statement creates a loop that will never terminate because the condition true is always true. As a result, the program will continuously execute the code inside the loop. To avoid infinite loops, we need to ensure that there is a condition inside the loop that will eventually become false or a break statement that will exit the loop.

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 :-

When we run this code we get NullPointerException. This exception is thrown when an application attempts to use null in a case where an object is required.

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 :-

The given code snippet will not compile successfully due to a data type mismatch error. The Java compiler will throw an error - incompatible types: String cannot be converted to double. Java enforces type safety to prevent errors at runtime, ensure data integrity.

Correct code :-

```
public class Main {

public static void main(String[] args) {

String num = "Hello";

System.out.println(num);

}

}
```

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 :-

The code will produce a compilation error due to an incompatible data types.error - incompatible types: possible lossy conversion from double to int.

Correct code :-

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

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 output is 2.0

To perform floating-point division and get a precise result with the decimal part (2.5), at least one of the operands must be of type double or float.

Correct code :-

```
public class Main {

public static void main(String[] args) {

int num = 10;

double result = (double) num / 4;

System.out.println(result);

}

}
```

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 result in a compilation error due to the use of the ** operator. Error - operator ** cannot be applied to int, int. The ** operator is not a valid operator in Java. Java does not support the ** operator for exponentiation or power calculations. Java supports basic arithmetic operators such as +, -, *, and / for addition, subtraction, multiplication, and division, respectively.

Correct code :-

```
public class Main {
```

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

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 the code is 20. Java follows specific rules for operator precedence, which determines the order in which operations are performed in an expression. The multiplication operator (*) has higher precedence than the addition operator (+).

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 :-

When we run this code, it will throw an ArithmeticException with the message - Exception in thread "main" java.lang.ArithmeticException: / by zero. Division by zero is mathematically undefined and is not allowed in Java. Specifically, the Java Virtual Machine (JVM) cannot handle division by zero in arithmetic operations involving integers.

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 :-

The code snippet has a syntax error due to a missing semicolon. In java the semicolon acts as a statement terminator, indicating the end of a complete instruction to the compiler.

Correct code :-

```
public class Main {

public static void main(String[] args) {

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

}

}
```

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 we attempt to compile this code, the Java compiler will generate an error due to the missing closing brace. To fix the error, we need to add the missing closing brace to properly terminate the main method and the Main class.

Correct code :-

```
public class Main {

public static void main(String[] args) {

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

}

}
```

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 code will result in a syntax error due to the incorrect placement of the displayMessage method. Specifically, the error occurs because you cannot declare a method within another method.

Correct code :-

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

```
        displayMessage();
    }

    static void displayMessage() {
        System.out.println("Message");
    }
}
```

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 :-

It gives the unexpected result because we have not written the break statement after each case. To avoid this we must write break statement after each case statement.

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 :-

It gives compilation error: The switch statement does not support double type. Switch expressions in Java must be of type byte, short, char, int, enum, String, or var. To make the code work, we need to use an integer type for the switch expression or cast the double value to an integer type.

Correct code :-

```
public class Switch {
public static void main(String[] args) {
int score = 85;
switch(score) {
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? 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 in a switch block must be unique. Having two identical case labels (case 5:) causes ambiguity, as the compiler cannot determine which block to execute. This results in a compilation error. To fix it, ensure all case labels are distinct.

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"

Ans :-

```
public class Score {
   public static void main(String args[]) {
      int marks = 65;
      if (marks >= 90) {
         System.out.println("A");
      } else if (marks >= 80 && marks <= 89) {
         System.out.println("B");
      } else if (marks >= 70 && marks <= 79) {
         System.out.println("C");
      } else if (marks >= 60 && marks <= 69) {
         System.out.println("D");
      } else if (marks < 60) {
         System.out.println("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 it is a weekday or weekend.

Ans :-

```
public class Daysofweek {
   public static void main(String args[]) {
      int n = 5;
      switch (n) {
         case 1:
            System.out.println("Monday");
            break;
         case 2:
            System.out.println("Tuesday");
            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("Enter correct number");
            break;
        }
    }
}
```

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

Ans :-

```
import java.util.*;


public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the first number");
        int n1 = sc.nextInt();
        System.out.println("Enter the second number");
        int n2 = sc.nextInt();
        System.out.println("Enter an operator");
        char operator = sc.next().charAt(0);
        switch (operator) {
            case '+':
                System.out.println("The result is " + (n1 + n2));
                break;
            case '-':
                System.out.println("The result is " + (n1 - n2));
                break;
```

```
        case '*':
            System.out.println("The result is " + (n1 * n2));
            break;
        case '/':
            if (n2 == 0) {
                System.out.println("division by zero is not allowed");
            } else {
                System.out.println("The result is " + (n1 / n2));
            }
            break;

        default:
            System.out.println("Enter correct operator");
            break;
    }
  }
}
```

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.

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

```
import java.util.Scanner;

public class Main {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double totalAmount = scanner.nextDouble();
        String membership = scanner.next().toLowerCase();
        double discount;
        if (totalAmount >= 1000) {
            discount = 20;
        } else if (totalAmount >= 500) {
            discount = 10;
        } else {
            discount = 5;
        }
        if (membership.equals("yes")) {
            discount += 5;
        }
        double discountAmount = (discount / 100) * totalAmount;
        double finalAmount = totalAmount - discountAmount;
        System.out.println("Discount: " + discount + "%");
        System.out.println("Discount Amount: Rs. " + discountAmount);
        System.out.println("Final Amount to Pay: Rs. " + finalAmount);
    }
}
```