**Assignment 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?

**error:**

* The main method should be public static void main(String[] args) instead of public void main(String[] args).
* Missing static will cause a runtime error because the JVM looks for public static void main(String[] args).

**Corrected 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?

**Error:**

- public access modifier is missing

- which will prevent the JVM from finding the main method at runtime.

**Corrected 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?

**Error:**

-Incorrect Return Type for main Method:

-The main method must have a void return type because the JVM does not expect it to return any value.

-main method has int as the return type, which will cause a runtime error since the JVM won’t recognize it as the entry point.

**Corrected 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?

**Compilation:**

The code compiles successfully because main() is a valid method declaration in Java.

**Execution:**

-The program will not run and will throw a runtime error, saying:

Error: Main method not found in class Main, please define the main method as:

public static void main(String[] args)

**Why is String[] args Needed?**

-JVM Entry Point Requirement:

-The Java Virtual Machine (JVM) requires the main method to have the signature

public static void main(String[] args)

-Without String[] args, the JVM does not recognize the method as the program’s entry point.

**Command-Line Arguments Support:**

String[] args allows users to pass command-line arguments to the program.

-------------------------------------------------------------------------------------------------------------------------

**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?

-Yes, we can have multiple main methods in Java.

**Observations:**

-Java allows multiple main methods.

-JVM always looks for public static void main(String[] args) as the entry point and ignores other overloaded versions.

-The overloaded main(int[] args) method will only execute if called explicitly from another method.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Error:**

-cannot find symbol int x = y + 10; ^ symbol: variable y location: class Main

**Why must variables be declared?**

-Variables must be declared before use in Java to avoid compilation errors.

-Always initialize variables before using them in expressions.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Error:**

incompatible types: String cannot be converted to int

int x = "Hello";

-Java enforces type safety to prevent assigning incompatible data types to variables.

-This helps catch errors early and ensures reliability in code execution.

-If you need to store a string, declare the variable as String instead of int.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Syntax Error**

-Missing Closing Parenthesis ) in System.out.println

- Missing Semicolon ;

Even after adding the missing parenthesis, you'd need a semicolon ; at the end of the statement.

**How do they affect compilation?**

Prevents Compilation:

Syntax errors stop the Java compiler from compiling the code.

The program won’t run until all syntax errors are fixed.

Error Cascading:

Sometimes, a single syntax error (like a missing parenthesis) can cause multiple compilation errors.

The compiler may misinterpret subsequent code, leading to confusing error messages.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Error :**

not a statement int class = 10; ^ error: ';' expected System.out.println(class);

-The main issue is the use of the **reserved keyword class** as a variable name.

-Reserved keywords cannot be used as identifiers in Java because they have predefined meanings in the language.

-Using them as variable names leads to compilation errors and ambiguity.

-------------------------------------------------------------------------------------------------------------------------

**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?

**Error :**

Getting compilation error

**Is Method Overloading Allowed?**

Yes, method overloading is allowed in Java.

Method Overloading occurs when multiple methods in the same class have the same name but different parameter lists.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Runtime Exception** : ArrayIndexOutOfBoundsException

Why Does This Exception Occur?

**1.Array Indexing Issue:** In Java, array index start at 0.

The array arr has 3 elements:

Index: 0 1 2

Value: 1 2 3

**Accessing an Invalid Index:**

The code tries to access arr[5], but the valid indices are only 0 to 2.

Index 5 is out of bounds, leading to a java.lang.ArrayIndexOutOfBoundsException.

-----------------------------------------------------------------------------------------------------------------------

**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?

**What Happens When You Run This Code?**

-The program will print "Infinite Loop" endlessly without stopping.

-You will need to manually terminate the program

**How to Avoid Infinite Loops?**

-Option 1: Add a Termination Condition

-Option 2: Use a break Statement

--------------------------------------------------------------------------------------------------------------------------

**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?

**Exception**: The java.lang.NullPointerException is thrown.

**Why Does This Exception Occur?**

-In Java, null represents the absence of an object reference.

-The variable str is declared but does not point to any actual String object (str = null).

-The code tries to execute str.length(), which invokes the length() method on the String object.

-Since str is null, there is no actual object to call length() on.

-This leads to a NullPointerException because Java can't dereference a null reference.

-------------------------------------------------------------------------------------------------------------------------

**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?

**Compilation Error:**

"incompatible types: String cannot be converted to double".

**Why does Java enforce data type constraints?**

-The variable num is declared as a double, which is a primitive data type in Java used to store floating-point numbers (e.g., 3.14, 7.0).

-The code tries to assign "Hello" (a String) to num, which causes a type mismatch.

-Java is a strongly typed language, which means that variables must be assigned values of compatible types.

-It doesn't allow implicit conversion between unrelated types (like String → double).

------------------------------------------------------------------------------------------------------------------------

**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?

**Error**: incompatible types: possible lossy conversion from double to int

**How Should You Handle Different Data Types in Operations?**

1. Use a double to Store the Result:

-Since one operand is a double, it's logical to store the result as a double to preserve precision.

-Cast double to int Before Addition (If You Want Integer Arithmetic):

--------------------------------------------------------------------------------------------------------------------------

**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?

**Answer**

expected output is 2.5, but actual output is 2.0

**Why Does This Happen?**

Integer Division:

Both num (10) and 4 are integers.

When divide two integers in Java (int / int), the result is also an integer, and any fractional part is truncated (not rounded).

10 / 4 results in 2, not 2.5.

**Assignment to double:**

The integer result 2 is then implicitly cast to double when assigned to result, resulting in 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?

**Error**: illegal start of expression

int result = a \*\* b;

**Why is the \*\* operator not valid in Java?**

The \*\* operator is not valid in Java

----------------------------------------------------------------------------------------------------------------------

**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?

**Output** : 20

How does operator precedence affect the result?

**Java's Arithmetic Operator Precedence (High to Low):**

1. Multiplication (\*)

2. Division (/)

3. Modulus (%)

4. Addition (+)

5. Subtraction (-)

**Evaluate Multiplication First** **(\*):** b \* 2 → 5 \* 2 = 10

**Then Addition (+):** a + (b \* 2) → 10 + 10 = 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);

}

}

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

**Runtime Exception** : Exception in thread "main" java.lang.ArithmeticException: / by zero

at Main.main(Main.java:5)

**Why Does Division by Zero Cause an Issue in Java?**

**Undefined Mathematical Operation**:

In mathematics, division by zero is undefined.

10 / 0 has no real value, which leads to an error.

Java throws an ArithmeticException when dividing an integer by zero to avoid invalid results and ensure program stability.

This is a runtime exception, meaning the code compiles successfully but fails during execution.

--------------------------------------------------------------------------------------------------------------------------

**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?

**Error : Missing Semicolon**

In Java, semicolons (;) are statement terminators.

Every complete statement must end with a semicolon to tell the compiler that the instruction is finished.

**Impact of Missing Semicolon:**

Without the semicolon, the Java compiler can't recognize where the statement ends, leading to a syntax error.

The compiler stops parsing the code and won't compile the program until the error is fixed.

------------------------------------------------------------------------------------------------------------------------

**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?

**Error :** reached end of file while parsing

**Role of Braces {} in Java:**

Java uses curly braces {} to define code blocks for classes, methods, loops, conditionals, etc.

Every opening brace { must have a corresponding closing brace }.

**Cause of the Error:**

The compiler encountered the end of the file before finding the closing brace for the main method and the Main class.

It leads to the reached end of file while parsing error, meaning the compiler was expecting more code (i.e., a closing brace) but reached the file's end.

----------------------------------------------------------------------------------------------------------------------

**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?

**Answer**

-Method Declarations Inside Methods Are Not Allowed

-In Java, methods cannot be declared inside other methods.

-The method displayMessage() is incorrectly declared inside main(), which causes the syntax error.

-------------------------------------------------------------------------------------------------------------------------

**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?

**Why Does This Happen?**

-Once a matching case is found (case 2 here), all subsequent cases (including case 3 and default) will execute, regardless of their values.

**Detailed Flow:**

value = 2 matches case 2.

It prints "Value is 2".

Since there’s no break, it continues executing:

Prints: "Value is 3" (from case 3)

Prints: "Default case" (from default)

**How can you prevent the program from executing the default case?**

Use break statements after each case to stop the flow

--------------------------------------------------------------------------------------------------------------------------

**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?

-When level = 1, it matches case 1 and prints "Level 1".

Without a break, the program continues executing the following cases sequentially:

Prints "Level 2" (from case 2)

Prints "Level 3" (from case 3)

Prints "Unknown level" (from default)

-break statements stop the flow after executing a matched case.

-Without them, Java assumes you want to execute all subsequent cases

**How to Fix the Code?**

Add break statements after each case.

**The Role of break:**

Without break, all subsequent cases will execute after a match.

--------------------------------------------------------------------------------------------------------------------------

**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?

-**double and float are not allowed** as switch expressions due to the potential for precision errors and lossy conversions.

**How Can You Modify the Code to Make It Work?**

**Option 1:** Use if-else Instead of switch

Since double can’t be used in a switch, convert the logic to use an if-else statement.

public class Switch {

public static void main(String[] args) {

double score = 85.0;

if (score == 100.0) {

System.out.println("Perfect score!");

} else if (score == 85.0) {

System.out.println("Great job!");

} else {

System.out.println("Keep trying!");

}

}

}

**Option 2: Use an int Instead of double in switch**

If precise floating-point comparison isn't necessary, you can cast the double to an int

public class Switch {

public static void main(String[] args) {

int score = (int)85.0; // Cast to int

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?

**Error:**

-duplicate case label case 5:

**Why Does This Error Occur?**

1. Duplicate Case Labels:

-In a switch statement, each case label must be unique.

-Having two case 5: labels creates ambiguity, as the compiler wouldn't know which block to execute if number is 5.

-This violates the syntax rules of Java, leading to a "duplicate case label" error.

Switch Case Matching Logic:

-When a switch statement runs, it compares the expression (number in this case) against each case label.

-It will execute the code under the first matching case and then exit the switch after hitting a break (if present).

H-aving duplicate cases makes this matching logic impossible for the compiler to resolve.

**What Happens When You Have Duplicate Case Labels?**

-Compilation fails due to the ambiguity.

-The program will not run until the duplicate case is removed or modified.

**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"

public class GradeClassification {

public static void main(String args[]){

int score = 78; //predefined score

if(score >= 90){

System.out.println("A");

}else if(score >=80 ){

System.out.println("B");

}else if(score >=70 ){

System.out.println("C");

}else if(score>=60 ){

System.out.println("D");

}else{

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.

public class DayOfWeek{

public static void main(String[] args){

int day = 7; //Predefined value

switch(day) {

case 1:

System.out.println("Monday");

System.out.println("It's a weekday");

break;

case 2:

System.out.println("Tuesday");

System.out.println("It's a Weekday");

break;

case 3:

System.out.println("Wednesday");

System.out.println("It's a weekday");

break;

case 4:

System.out.println("Thursday");

System.out.println("It's a Weekday");

break;

case 5:

System.out.println("Friday");

System.out.println("It's a weekday");

break;

case 6:

System.out.println("Saturday");

System.out.println("It's a Weekend");

break;

case 7:

System.out.println("Sunday");

System.out.println("It's a Weekend");

break;

default:

System.out.println("Invalid Input");

}

}

}

----------------------------------------------------------------------------------------------------------------------

**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.

public class SimpleCalculator{

public static void main(String args[]){

double num1 = 10.5;

double num2 = 0;

char operator = '+';

double result;

switch(operator) {

case '+':

result = num1 + num2;

System.out.println("result : " +result);

break;

case '-':

result = num1 - num2;

System.out.println("result : " +result);

break;

case '\*':

result = num1 \* num2;

System.out.println("result : " +result);

break;

case '/':

if (num2 == 0) {

System.out.println("Error: Division by zero is not allowed.");

} else {

result = num1 / num2;

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

}

break;

default:

System.out.println("Error: Invalid operator.");

}

}

}

--------------------------------------------------------------------------------------------------------------------------

**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%.

public class DiscountCalculation {

public static void main(String[] args) {

double totalPurchase = 750; // Predefined total purchase amount

boolean hasMembershipCard = true; // Predefined membership status

double discount;

if (totalPurchase >= 1000) {

discount = 20;

} else if (totalPurchase >= 500) {

discount = 10;

} else {

discount = 5;

}

if (hasMembershipCard) {

discount += 5;

}

double discountAmount = (discount / 100) \* totalPurchase;

double finalAmount = totalPurchase - discountAmount;

System.out.println("Total Purchase: Rs." + totalPurchase);

System.out.println("Discount Applied: " + discount + "%");

System.out.println("Final Amount: Rs." + finalAmount);

}

}

--------------------------------------------------------------------------------------------------------------------------

**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.

public class StudentPassFail {

public static void main(String[] args) {

int subject1 = 35; // Predefined grade for subject 1

int subject2 = 50; // Predefined grade for subject 2

int subject3 = 30; // Predefined grade for subject 3

int failCount = 0;

if (subject1 <= 40) {

failCount++;

}

if (subject2 <= 40) {

failCount++;

}

if (subject3 <= 40) {

failCount++;

}

if (failCount == 0) {

System.out.println("Student passes.");

} else {

System.out.println("Student fails in " + failCount + " subject(s).");

}

}

}