**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 : // Static are missing**

**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 : // Public are missing**

**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 : // void return type is not there in main method void tells java the main method won’t return the value**

**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 : // argument and string not declare**

**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 : // Yes we use multiple main methods -> Method overloading**

**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: // y is not declared java needs to declare data type to execute the program**

**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: // There should be String variable 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?**

**Ans: // Parentheses missing**

**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: // Class is a reserve keyword**

**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: // non static method cannot be referenced from static**

**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: // It shows array out of bound exception array size is 0,1,2 only but we accessing index 5**

**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: // Yes infinite loop occur**

**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: //** In this code, the variable str is declared and explicitly set to null.
* The line System.out.println(str.length()); attempts to call the length() method on the str object.
* However, since str is null and does not reference any actual String object, there is no object to call the length() method on.
* This results in the NullPointerException being thrown at runtime

**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: // String cannot be converted into a 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?**

**Ans: // incompatible types lossy conversion**

**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: // 2.0 is the output**

**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: // Invalid Operator(\*\*) Java does not support exponention to perform exponention in java Math.pow() use**

**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: // b \* 2 is evaluated first: 5 \* 2 = 10.

### Then, a + 10 is evaluated: 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?

**Ans:// Mathematical Reason**: In mathematics, division by zero is undefined because there is no number that, when multiplied by zero, gives a non-zero number. This is why any attempt to divide by zero leads to an error.

**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://** In Java, every statement must end with a semicolon (;). The semicolon acts as a statement terminator, telling the compiler where one statement ends and the next begins.

**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: // error: illegal start of expression static void displayMessage()

**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: //** In Java, methods cannot be declared inside other methods. All methods must be declared at the class level, directly inside the class body, but not inside other methods or constructors.

**Snippet23:**

publicclassConfusion{

publicstaticvoidmain(String[]args){ int value = 2;

switch(value){ case 1:

System.out.println("Valueis1");

case 2:

System.out.println("Valueis2");

case 3:

System.out.println("Valueis3"); default:

System.out.println("Defaultcase");

}

}

}

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

* **Explanation:**
* **Switch Case Fall-Through**: In Java, once a matching case is found in a switch statement, all the subsequent cases are executed until a break statement or the end of the switch block is encountered. This is known as "fall-through."
* **Why Default Case Prints**: Since there is no break statement after case 2, the program continues to execute case 3 and then the default case, printing all of them.
* **How to Prevent Fall-Through:**
* To prevent the program from executing subsequent cases after a match, you should add break statements at the end of each case:

**Snippet24:**

publicclassMissingBreakCase{

publicstaticvoidmain(String[]args){ int level = 1;

switch(level){ case 1:

System.out.println("Level1");

A close up of a logo

Description automatically generatedcase 2:

System.out.println("Level2");

case 3:

System.out.println("Level3"); default:

System.out.println("Unknownlevel");

}

}

}

•**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?

* **Switch Case Fall-Through**: In Java, when a switch statement executes, it starts at the matching case and continues executing subsequent cases until it encounters a break statement or reaches the end of the switch block. This is known as "fall-through."

**Role of the break Statement:**

* **Preventing Fall-Through**: The break statement is used to terminate the switch case after the matching case has been executed. When the break statement is encountered, the program exits the switch block, preventing any further cases from being executed.
* **Correct Usage**: To stop the fall-through and ensure that only the matching case is executed, you should add break statements at the end of each case.

**Snippet25:**

publicclassSwitch{

publicstaticvoidmain(String[]args){ double score = 85.0;

switch(score){ case 100:

System.out.println("Perfectscore!"); break;

case 85:

System.out.println("Greatjob!"); break;

default:

System.out.println("Keeptrying!");

}

}

}

•**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 the switch statement in Java does not support double(orfloat)values.Theswitchstatementcanonlybeusedwiththefollowingtypes: byte, short, int, char, enum, String and var.

-To make the code work, you can either convert the double value to an int, use a different type supported by switch, or use an if-else structure instead.

**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:**Thecompilercomplainaboutduplicatecaselabelsbecause:

InJava,withinaswitchstatement,eachcaselabelmustbeuniquewithintheswitchblock. 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.