**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: Snippet1.java:1: error: class Main is public, should be declared in a file named Main.java**

**public class Main {**

**^**

**1 error**

**Error: main method must be static.   
  
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: Error: Could not find or load main class Snippet2**

**Caused by: java.lang.ClassNotFoundException: Snippet2  
Error: The program compiles successfully but does not run.**

**The main method must be declared as public static void main(String[] args), but in this snippet, it is only static void main(String[] args). Since it's not public, the JVM cannot find the entry point to execute the program.**

**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: Snippet3.java:5: error: reached end of file while parsing**

**}**

**^**

**1 error  
Error: main method must return void.  
The main method in Java must have a return type of void because it serves as the entry point for the JVM and does not return any 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: Error: Main method not found in class Main, please define the main method as: public static void main(String[] args)  
Reason: String[] args is needed to accept command-line arguments. Without it, the JVM cannot recognize the method as the entry point.**

**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, multiple main methods can exist (overloaded), but only the one with String[] args is executed by the JVM unless the overloaded method is explicitly called within the program.  
  
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: Error: Compilation error – "Cannot find symbol: variable y."**

**Reason: The variable y is used before declaration. In Java, all variables must be declared before use to allocate memory and define their type.  
  
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: Error: Incompatible types – String cannot be assigned to int.**

**Reason: Java enforces type safety to prevent runtime errors by ensuring variables hold only values of their declared type.  
  
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: Errors: Missing closing parenthesis “)” and semicolon “;”.**

**Effect: The code will not compile due to syntax errors. The compiler expects proper closing of method calls and statements.  
  
Snippet 9:**

public class Main {

public static void main(String[] args) {

int class = 10;

System.out.println(class);

}

} a

** What error occurs? Why can't reserved keywords be used as identifiers?  
  
ANS: Error: class is a reserved keyword and cannot be used as a variable name.**

**Reason: Java reserves certain words for language constructs, preventing their use as identifiers to avoid ambiguity and conflicts in code execution.  
  
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: Error: display() and display(5) cannot be called directly inside main() because they are non-static methods.**

**Fix: Create an object of Main to call display().**

**Method Overloading: Yes, method overloading is allowed in Java.  
  
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: Exception: ArrayIndexOutOfBoundsException**

**Reason: The array has only 3 elements (indices 0 to 2), but index 5 is accessed, which is out of bounds.  
  
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: Output: The program prints "Infinite Loop" continuously.**

**Avoidance: Use a termination condition inside the loop, like a break statement or a proper condition check.  
  
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: Exception: NullPointerException**

**Reason: str is null, so calling .length() on it causes an error.  
  
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: Error: Incompatible types: String cannot be converted to double.**

**Reason: Java enforces strict type safety to prevent invalid assignments and ensure reliable code execution.  
  
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: Error: Incompatible types: possible lossy conversion from double to int.**

**Solution: Use explicit casting (int result = (int) (num1 + num2);) or store the result in a double variable.  
  
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: Output: 2.0**

**Reason: Integer division (10 / 4) results in 2, and then it is stored as 2.0 in double.**

**Fix: Use double division: double result = num / 4.0; to get 2.5.  
  
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: Snippet17.java:5: error: illegal start of expression**

**int result = a \*\* b;**

**^**

**1 error  
Error: Compilation error – \*\* is not a valid operator in Java.**

**Reason: Java does not support \*\* for exponentiation. Use Math.pow(a, b) instead.  
  
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: Output: 20**

**Reason: Multiplication (\*) has higher precedence than addition (+), so b \* 2 is evaluated first (5 \* 2 = 10), then added to a (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: Exception: java.lang.ArithmeticException: / by zero**

**Reason: In Java, dividing an integer by zero is undefined and causes an ArithmeticException at runtime.  
  
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: Error: Syntax error: missing ';' before '}'**

**Reason: The missing semicolon (;) after "Hello, World" causes a compilation error because Java requires statements to end with a semicolon.**

**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: Syntax error: reached end of file while parsing**

**Reason: The missing closing brace (}) causes a compilation error because Java expects properly matched braces to define code blocks.  
  
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: Error: Illegal start of expression**

**Reason: In Java, methods cannot be declared inside another method. Methods must be defined at the class level, not inside main().  
  
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: Reason: The switch statement is missing break statements. Without break, execution falls through to the next case, including default.**

**Fix: Add break; after each case to stop execution:  
  
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: Reason: The switch statement lacks break statements, causing "fall-through" behavior. When level is 1, execution continues through all subsequent cases, printing all messages.**

**Fix: Add break; after each case to stop execution.**

**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: Error: The code does not compile because switch expressions must use integer types (int, char, byte, short, enum, or String). A double value is not allowed.**

**Fix: Convert score to an int or use an if-else statement instead  
int score = 85; // Convert 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!");**

**}  
  
Or use an if-else statement for double values:  
  
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!");  
  
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: Error: The compiler throws an error due to duplicate case labels (case 5: appears twice). Each case label in a switch statement must be unique.**

**The switch statement needs distinct cases to determine execution flow. Duplicate labels cause ambiguity, making it impossible for Java to decide which case to execute.**

**Fix: Remove or modify the duplicate case**