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:

* Error: No main method found.
* The main method must be public static void main(String[] args), but here it is not static, so Java doesn’t recognize it as the entry point.
* Solution:

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:

* Compiles but doesn’t run.

The main method is static but not public, so Java fails to find the valid entry point.

* Solution:

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:

* Error: main method must return void.Java requires void for main since it is the entry point and doesn't expect a return value.
* Solution:

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:

* Compiles but doesn’t run. The method signature is incorrect because it is missing String[] args. Java needs this parameter to recognize the main method.
* Solution:

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:

* Error: Overloaded main methods are allowed.
* Observation: Only public static void main(String[] args) is the entry point. Other main methods must be called explicitly.
* Solution:

public class Main {

public static void main(String[] args) {

System.out.println("Main method with String[] args");

main(new int[]{}); // Explicitly calling overloaded main

}

public static void main(int[] args) {

System.out.println("Overloaded main method with int[] args");

}

}

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: The variable y is not declared before it is used.
* Reason: Java requires variables to be declared before use to define their data types, ensuring that the compiler knows what type of value it is working with.
* Solution:

public class Main {

public static void main(String[] args) {

int y=0;

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:

* Error: int x = "Hello";
* Reason: Java enforces type safety, meaning that it does not allow assigning a String value to an int variable because they are incompatible types.
* Solution:

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:

* Errors: The closing parenthesis ) is missing in System.out.println("Hello, World!".
* Effect: This causes a syntax error because the method call is not properly terminated.
* Solution:

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:

* Error: The identifier class cannot be used as a variable name because class is a reserved keyword in Java.
* Reason: Reserved keywords have predefined meanings in the language and cannot be used as identifiers.
* Solution:

public class Main {

public static void main(String[] args) {

int c = 10;

System.out.println(c);

}

}

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:

* Result: Compilation fails because you cannot call the non-static method display() from a static context (in the main method).
* Solution: To use the display() method, you need to create an instance of the Main class or make display() method static.

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) {

Main obj = new Main();

obj.display();

obj.display(5);

}

}

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: Accessing arr[5] in an array of size 3 exceeds the valid index range (0, 1, 2), leading to an ArrayIndexOutOfBoundsException.
* Solution:

public class Main {

public static void main(String[] args) {

int[] arr = {1, 2, 3};

System.out.println(arr[2]);

}

}

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:

* Result: The program will run indefinitely, printing "Infinite Loop".
* Solution: To avoid infinite loops, ensure there is a valid exit condition or use a break statement.

public class Main {

public static void main(String[] args) {

int count = 0;

while (true) {

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

count++;

if (count == 5) {

break;

}

}

}

}

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: Attempting to call .length() on a null object will result in a NullPointerException.
* Solution:

public class Main {

public static void main(String[] args) {

String str = null;

if (str != null) {

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

} else {

System.out.println("String is null, cannot get length.");

}

}

}

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: double num = "Hello";
* Reason: Java requires data type consistency. A String cannot be assigned to a double variable, which enforces type safety.
* Solution:

public class Main {

public static void main(String[] args) {

double num = 10.5;

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:

* Error: int result = num1 + num2;
* Reason: You cannot directly add an int and a double without type casting. Java does not implicitly convert double to int.
* Solution: Explicitly cast the double to an int or change the result variable to double.

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:

* Result: The output will be 2.5 because 10 / 4 results in 2.5 when the result is stored in a double.
* Solution:

public class Main {

public static void main(String[] args) {

int num = 10;

double result = num / 4.0;

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:

* Error: \*\* operator is not valid in Java.
* Reason: Java does not support exponentiation with \*\*; instead
* Solution:

public class Main {

public static void main(String[] args) {

int a = 10;

int b=5;

int result = 1;

for (int i = 0; i < b; i++)

{

result = result \*a;

}

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:

* Output: 20
* Reason: Operator precedence dictates that multiplication (b \* 2) is performed before addition (a + result).
* Solution:

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

int result = a + (b \* 2);

System.out.println(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:

* Exception: ArithmeticException for division by zero.
* Reason: Java throws an exception when attempting to divide by zero.
* Solution:

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 0;

if (b != 0) {

int result = a / b;

System.out.println(result);

} else {

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

}

}

}

* 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: Missing semicolon after the print statement.
* Effect: The missing semicolon causes a compilation error because every statement in Java must end with a semicolon.
* Solution:

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:

* Error: Mismatched braces.
* Effect: The missing closing brace causes a syntax error and prevents the program from compiling.
* Solution:

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:  
  Error: Methods cannot be declared inside other methods.
* Reason: The method displayMessage() cannot be declared inside the main() method.
* Solution:

public class Main {

public static void displayMessage() {

System.out.println("Message");

}

public static void main(String[] args) {

displayMessage();

}

}

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:

* Issue: The lack of break statements causes all subsequent case statements to execute, leading to "Value is 2", "Value is 3", and "Default case" being printed.
* Solution: Add break statements after each case to prevent fall-through behavior.

public class Confusion {

public static void main(String[] args) {

int value = 2;

switch(value) {

case 1:

System.out.println("Value is 1");

break;

case 2:

System.out.println("Value is 2");

break;

case 3:

System.out.println("Value is 3");

break;

default:

System.out.println("Default case");

}

}

}

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:  
  Issue: The program prints "Level 1", "Level 2", "Level 3", and "Unknown level" because of the missing break statements.
* Solution: Add break statements to prevent fall-through.  
  public class MissingBreakCase {

public static void main(String[] args) {

int level = 1;

switch(level) {

case 1:

System.out.println("Level 1");

break;

case 2:

System.out.println("Level 2");

break;

case 3:

System.out.println("Level 3");

break;

default:

System.out.println("Unknown level");

}

}

}

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: You cannot use a double in a switch statement.
* Reason: Java supports only byte, short, char, and int (or their wrapper classes) in switch expressions.
* Solution: Change score to an int or char type to make it compatible with switch.

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:

* Error: Duplicate case labels (case 5).
* Reason: Case labels must be unique within a switch statement.
* Solution: Remove the duplicate case 5 or assign different values to the cases.

public class SwitchExample {

public static void main(String[] args) {

int number = 5;

switch(number) {

case 5:

System.out.println("Number is 5");

break;

default:

System.out.println("This is the default case");

}

}

}