## ASSIGNMENT 2##

**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 not found in class

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 :** we will get errorMain method not found in class

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 :** Main method not found in class jvm look for the exact same method which is

*public static void main(String[] args) { }*

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 :** Main method not found in class , String[] args is accepts the multiple argument we can pass while before start the method

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** : yes we can have multiple main method but jvm only execute one main which is

public static void main(String[] args) { System.out.println("Main method with String[] 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?**

**Answer** : cannot find the symbol y in main class

For type safety and memory allocation and error preventation variable must be decalre

**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 converted to int

For code compilation , memory management , code clarity java enforce the type safety

**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 : There is a missing closing parenthesis ) at the end of the System.out.println("Hello, World!") statement.

The Java compiler requires that all statements end with a semicolon and that all parentheses are correctly closed. The missing closing parenthesis and semicolon will cause the compiler to fail with a syntax error.

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: 'class' cannot be used as an identifier

Compiler Confusion, Readability Consistency that’s Why can't reserved keywords be used as identifiers

**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**: non-static method display() cannot be referenced from a static context

**Method Overloading:** Allowed in Java; methods with the same name but different parameters can coexist.

**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?**

**Error** **with explanation**: index out bound exception we are trying to access 5 element but array size is 3

**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 :** Console will print “infinite loop” without stopping for stopping it we should add break statement and pass such condition which is satisfy rather than direct passing of true using variable we need to pass

**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?**

**Error : we will get the NullpointerException**

The variable str is assigned the value null, meaning it does not reference any actual String object.

**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?**

**Answer:Error:** incompatible types: String cannot be converted to double

For code compilation , memory management , code clarity java enforce the type safety

**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

Type Promotion, Casting, Use Appropriate Data Types Avoid Loss of Precision that’s why we need to handle different type in operations

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?**

**Output is => 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:** cannot find symbol symbol: method \*\*(int,int)

Java's syntax does not include the \*\* operator for exponentiation

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 is => 20**

In Java, operators have different levels of precedence that determine the order in which operations are performed. 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 :**

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

In Java, dividing an integer by zero is not allowed and results in an ArithmeticException. This is because division by zero is mathematically undefined, and Java's arithmetic operations follow this rule to avoid undefined behavior.

**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: ';' expected**

n Java, each statement must 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?**

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

Missing closing brace for the method and class definitions.

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?**

**Error**: illegal start of expression

**No:** In Java, methods cannot be declared inside other methods.

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 :** To prevent the program from executing the default case and any other cases after the matched case, you need to add break statements at the end of each case block

**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?

**Answer :** The absence of break statements causes the program to fall through from the matched case to all subsequent cases and the default case. The break statement prevents fall-through behavior, ensuring that only the code for the matched case executes.

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?**

**Error:** switch expression must be of type int, char, enum, or String

The switch statement expects the expression to be of one of the allowed types, and double is not compatible with this requirement. Thus, the code fails to compile.

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?**

**Answer:** **Error:** duplicate case label

When you have two identical case labels, the compiler cannot differentiate between them, making it unclear which block of code should execute for the matching case. This duplication results in a conflict because a switch statement expects each case to have a distinct value to properly route the flow of execution.