Assignment No - 2

Snippet 1:

public class Main {

public void main(String[] args) {

System.out.println("Hello, World!");

}

}

The provided Java code has an incorrect main method definition. The main method should be public static void main(String[] args), but in this snippet, the static keyword is missing.

Error Message When Compiling and Running the Code

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

public static void main(String[] args)

Fixed Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Added the static keyword to the main method.

Output:

Hello, World!

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

Snippet 2:

public class Main{

static void main(String[] args){

System.out.println("Hello, World!");

}

}

The provided Java code has an incorrect main method definition. The main method should be public static void main(String[] args), but in this snippet, the public access modifier is missing.

Error Message When Compiling and Running the Code

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

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

Fixed Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Added the public access modifier to the main method.

Output:

Hello, world!

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

Sniipet 3

public class Main {

public static int main(String[] args) {

System.out.println("Hello, World!");

return 0;

}

}

The provided Java code has an incorrect main method definition. The return type of main should be void, but in this snippet, it is int, which is not allowed.

The JVM starts execution from main and does not expect a return value.

Java uses System.exit(int status) for termination instead of returning an integer.

Error Message When Compiling and Running the Code

Error: Main method must return a value of type void in class Main, please

define the main method as:

public static void main(String[] args)

Fixed Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Changed the return type from int to void.

Output:

Hello, World!

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

Snippet 4:

public class Main {

public static void main() {

System.out.println("Hello, World!");

}

}

The provided Java code has an incorrect main method definition. The main method should be public static void main(String[] args), but in this snippet, it does not accept String[] args as a parameter

JVM Entry Point: The JVM looks for public static void main(String[] args).

Command-line Arguments: Allows passing arguments when running the program

Error Message When Compiling and Running the Code

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

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

Fixed Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Output

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");

}

}

Java allows method overloading, meaning you can define multiple methods with the same name but different parameter lists, including the main method.

The code compiles successfully because Java supports method overloading.

Output

Main method with String[] args

The overloaded main(int[] args) method is not executed automatically.

------------------------------------------------------------------------------------------------------------------ Snippet 6

public class Main {

public static void main(String[] args) {

int x = y + 10;

System.out.println(x);

}

}

The given Java code tries to use the variable y before declaring it. This results in a compilation error because Java requires all variables to be declared before use.

Error Message When Compiling the Code

Main.java:3: error: cannot find symbol

int x = y + 10;

^

symbol: variable y

location: class Main

1 error

Corrected Code

public class Main {

public static void main(String[] args) {

int y = 5;

int x = y + 10;

System.out.println(x);

}

}

Output

15

Java is a statically-typed language - All variables must have a type defined before use.

Ensures code clarity and prevents errors - Prevents accidental use of undefined variables.

Helps the compiler allocate memory correctly - The compiler needs to know the variable's type for proper memory allocation.

---------------------------------------------------------------------------------------------------------------------Snippet 7

public class Main {

public static void main(String[] args) {

int x = "Hello";

System.out.println(x);

}

}

Compilation Error

error: incompatible types: String cannot be converted to int

int x = "Hello";

^

1 error

Java does not allow assigning a String to an int variable because they are different data types.

It ensures variables store only valid data types, prevents runtime errors, and makes code more reliable.

Corrected Code

public class Main {

public static void main(String[] args) {

String x = "Hello";

System.out.println(x);

}

}

Output

Hello

---------------------------------------------------------------------------------------------------------------------Snippet 8

public class Main{

public static void main(String[] args){

System.out.println("Hello, World!"

}

}

Compilation Error

error: ')' expected

System.out.println("Hello, World!"

A closing parenthesis ) is missing before the closing curly brace } in System.out.println("Hello, World!".

A closing parenthesis ) is missing before the closing curly brace } in System.out.println("Hello, World!".

Corrected Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Output

Hello, World!

---------------------------------------------------------------------------------------------------------------------Snippet 9

public class Main{

public static void main(String[] args){

int class = 10;

System.out.println(class);

}

}

Compilation Error

Error: not a statement

int class = 10;

^

The word class is a reserved keyword in Java, so it cannot be used as a variable name.

Java uses keywords for specific language features, and allowing them as variable names would cause confusion and syntax errors.

Corrected Code

public class Main {

public static void main(String[] args) {

int number = 10;

System.out.println(number);

}

}

Output

10

---------------------------------------------------------------------------------------------------------------------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);

}

}

Compilation Error

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

display();

^

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

display(5);

^

The methods display() and display(int num) are non-static, meaning they belong to an instance of the class. However, they are being called inside main(), which is static and does not have access to instance methods directly.

Method overloading is allowed in Java. It allows multiple methods with the same name but different parameters. However, instance methods must be called using an object.

Corrected Code

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);

}

}

Output

No parameters

With parameter: 5

---------------------------------------------------------------------------------------------------------------------Snippet 11

public class Main {

public static void main(String[] args) {

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

System.out.println(arr[5]);

}

}

Runtime Exception

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

at Main.main(Main.java:4)

The array arr has only 3 elements (arr[0], arr[1], arr[2]).

Trying to access arr[5] is out of bounds, causing an ArrayIndexOutOfBoundsException.

Corrected Code

public class Main {

public static void main(String[] args) {

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

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

}

}

Output

3

---------------------------------------------------------------------------------------------------------------------Snippet 12

public class Main {

public static void main(String[] args) {

while (true) {

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

}

}

}

The program enters an infinite loop and keeps printing "Infinite Loop" continuously.

It will never stop unless manually terminated (Ctrl + C in terminal or stopping the program in an IDE).

Use a condition that eventually becomes false.

Add a break statement inside the loop to exit when needed.

Corrected Code

public class Main {

public static void main(String[] args) {

int count = 0;

while (count < 5) {

System.out.println("Loop iteration: " + count);

count++;

}

}

}

Output

Loop iteration: 0

Loop iteration: 1

Loop iteration: 2

Loop iteration: 3

Loop iteration: 4

---------------------------------------------------------------------------------------------------------------------Snippet 13

public class Main {

public static void main(String[] args) {

String str = null;

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

}

}

Exception Thrown

Exception in thread "main" java.lang.NullPointerException

at Main.main(Main.java:4)

The variable str is assigned null, meaning it does not reference any object.

Calling str.length() tries to access a method on a null value, which is not allowed, leading to a NullPointerException.

Ensure str is initialized before calling methods on it.

Add a null check before accessing properties.

Corrected Code:

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");

}

}

}

Output

String is null

---------------------------------------------------------------------------------------------------------------------Snippet 14

public class Main {

public static void main(String[] args) {

double num = "Hello";

System.out.println(num);

}

}

Compilation Error

Main.java:3: error: incompatible types: String cannot be converted to double

double num = "Hello";

^

1 error

The variable num is declared as a double (a numeric type).

Assigning a String ("Hello") to a double is not allowed because they are incompatible data types.

Java is a strongly typed language, meaning variables must store only values matching their declared type.

This prevents unexpected behavior, type mismatches, and runtime errors.

Corrected Code

public class Main {

public static void main(String[] args) {

double num = 10.5;

System.out.println(num);

}

}

Output

10.5

---------------------------------------------------------------------------------------------------------------------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);

}

}

Compilation Error

Main.java:4: error: incompatible types: possible lossy conversion from double to int

int result = num1 + num2;

^

1 error

num1 is an int, and num2 is a double.

The result of num1 + num2 is a double, but it is being stored in an int variable.

Java does not allow implicit conversion from double to int because it may result in data loss.

Use explicit casting if you intentionally want to store a double in an int.

Store the result in a double to avoid precision loss.

Corrected Code

public class Main {

public static void main(String[] args) {

int num1 = 10;

double num2 = 5.5;

double result = num1 + num2;

System.out.println(result);

int intResult = (int) (num1 + num2);

System.out.println(intResult);

}

}

Output

15.5

15

---------------------------------------------------------------------------------------------------------------------Snippet 16

public class Main {

public static void main(String[] args) {

int num = 10;

double result = num / 4;

System.out.println(result);

}

}

Output of the Code

2

num / 4 performs integer division because both num (int) and 4 (int) are integers.

In integer division, the decimal part is truncated, so 10 / 4 results in 2.

The result (2) is then stored in result (double), but since the division already removed the decimal, the output is 2.0 instead of 2.5.

Convert one of the numbers to double to perform floating-point division.

Corrected Code

public class Main {

public static void main(String[] args) {

int num = 10;

double result = num / 4.0;

System.out.println(result);

}

}

Expected Output

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);

}

}

Compilation Error

Main.java:5: error: illegal start of expression

int result = a \*\* b; ^

1 error

Java does not support the \*\* operator for exponentiation (unlike Python). The \*\* operator is not a valid arithmetic operator in Java, causing a syntax error.

Use Math.pow(base, exponent) from the Math class.

Since Math.pow() returns a double, store the result in a double variable or cast it if needed.

Corrected Code

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

double result = Math.pow(a, b);

System.out.println(result);

}

}

Output

100000.0

------------------------------------------------------------------------------------------------------------------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);

}

}

Output of the Code

20

Java follows operator precedence, meaning some operations are performed before others.

The multiplication (\*) operator has higher precedence than addition (+).

Use parentheses () to override default precedence.

Example: If you want addition first, write (a + b) \* 2.

Modified Code

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

int result1 = a + b \* 2;

int result2 = (a + b) \* 2;

System.out.println(result1);

System.out.println(result2);

}

}

Expected Output

20

30

------------------------------------------------------------------------------------------------------------------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);

}

}

Runtime Exception Thrown

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

at Main.main(Main.java:5)

In Java, integer division by zero is not allowed, leading to an ArithmeticException.

int result = a / b; tries to divide 10 by 0, which is undefined in mathematics.

Java does not handle integer division by zero gracefully and immediately throws an exception.

Check if the denominator is zero before performing the division.

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: " + result);

} else {

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

}

}

}

Output

Error: Division by zero is not allowed.

------------------------------------------------------------------------------------------------------------------Snippet 20

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World")

}

}

Syntax Error

Main.java:3: error: ';' expected

System.out.println("Hello, World")

^

1 error

The statement System.out.println("Hello, World") is missing a semicolon (;) at the end.

In Java, every statement must end with a semicolon (;) to indicate the end of the instruction.

The compiler expects a semicolon and throws a syntax error when it is missing.

Corrected Code:

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World");

}

}

Output

Hello, World

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

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

// Missing closing brace here

}

Compilation Error

Main.java:5: error: reached end of file while parsing

}

^

1 error

The code is missing a closing brace (}) at the end, which is required to close the class Main definition.

Java uses curly braces ({}) to define code blocks, and mismatched braces cause the compiler to reach the end of the file unexpectedly.

Corrected Code

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

Output

Hello, World!

------------------------------------------------------------------------------------------------------------------Snippet 22

public class Main {

public static void main(String[] args) {

static void displayMessage() {

System.out.println("Message");

}

}

}

Compilation Error

Main.java:3: error: illegal start of expression

static void displayMessage() {

^

1 error

In Java, methods cannot be declared inside other methods.

The displayMessage() method is inside the main() method, which is not allowed.

Methods must be declared inside a class, but outside other methods.

Corrected Code

public class Main {

static void displayMessage() {

System.out.println("Message");

}

public static void main(String[] args) {

displayMessage();

}

}

Output

Message

------------------------------------------------------------------------------------------------------------------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");

}

}

}

Issue Observed

Value is 2

Value is 3

Default case

In Java, switch statements execute fall-through behavior when break statements are missing.

Since there is no break after case 2, execution continues to case 3 and then to default.

Use break; to stop execution after each case if you don’t want fall-through behavior.

Corrected Code

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");

}

}

}

Output

Value is 2

------------------------------------------------------------------------------------------------------------------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");

}

}

}

Issue Observed

Level 1

Level 2

Level 3

Unknown level

The switch statement in Java follows fall-through behavior when break statements are missing.

Since case 1 has no break, execution continues through case 2, case 3, and default.

The break statement stops further execution inside a switch case.

Without break, Java continues executing all subsequent cases, regardless of their condition.

Corrected Code

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");

}

}

}

Output

Level 1

------------------------------------------------------------------------------------------------------------------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!");

}

}

}

Compilation Error

Main.java:4: error: incompatible types: possible lossy conversion from double to int

switch(score) {

^

1 error

The switch statement only supports specific data types:

byte, short, char, int

String, enum, Character, Byte, Short, Integer

Floating-point types (double, float) are not allowed in switch expressions.

Java requires exact matching of case values; since score is a double, the compiler throws an error.

Use an int instead of double

Use if-else instead of switch for floating-point comparisons

Corrected Code

public class SwitchExample {

public static void main(String[] args) {

int score = 85;t

switch(score) {

case 100:

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

break;

case 85:

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

break;

default:

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

}

}

}

Output

Great job!

------------------------------------------------------------------------------------------------------------------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");

}

}

}

Compilation Error

Main.java:7: error: duplicate case label

case 5:

^

1 error

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

Having duplicate case labels (case 5: appears twice) causes a compilation error.

Java does not allow multiple cases with the same constant value because it wouldn't know which one to execute.

Remove the duplicate case 5: or combine their logic.

Corrected Code

public class Switch {

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");

}

}

}

Output

Number is 5