*CDAC Mumbai*

*Lab Assignment*

*Section 1: Error-Driven Learning in Java*

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

   public static void main(String[] args).

**Corrected code:**

Java

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-**Hello, World!

This is because the main method in Java must have the following signature:

Java

public static void main(String[] args)

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-The error encountered in the code snippet is:

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)

Reason:

The main method in Java is a special method that serves as the entry point for a Java application. By convention, it is declared as public static void main(String[] args). The void keyword indicates that the main method 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)

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

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

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, you can have multiple main methods in a Java class, but only one of them can be declared as public static void main(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?

ANS-Main.java:3: error: cannot find symbol

    int x = y + 10;

            ^

  symbol:   variable y

  location: class Main

1 error

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

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-** Main.java:3: error: incompatible types: String cannot be converted to int

    int x = "Hello";

**^**

compilation error: Incompatible types: String cannot be converted to int.

Reason:

Java is a strongly typed language, which means that every variable must have a specific data type. The data type of a variable determines the range of values it can hold and the operations that can be performed on it.

In the given code snippet, the variable x is declared as an int, which means it can only hold integer values. However, you are trying to assign a String value ("Hello") to x. This is incompatible because String and int are different data types.

Why Java enforces type safety:

Java enforces type safety to prevent errors and make code more reliable. By ensuring that variables are used correctly, Java can help to avoid common programming mistakes such as:

* Arithmetic errors: If you try to perform arithmetic operations on variables of incompatible types, Java will generate a compilation error.
* Null pointer exceptions: If you try to access a member of a null object, Java will throw a NullPointerException.
* -invalid index, Java will throw an ArrayIndexOutOfBoundsException.
* Type conversion errors: If you try to convert a value of one type to another incompatible type, Java may generate a runtime error or produce unexpected results.

By enforcing type safety, Java helps to make code more robust and easier to maintain.

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

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-** Syntax errors are present in the code snippet. The missing semicolon at the end of the System.out.println("Hello, World!"); statement is the error. Semicolons are required to terminate statements in Java.

Impact on Compilation:

The compiler will encounter this error and fail to compile the code. It will generate an error message indicating that a semicolon is missing. The compilation process will halt until the syntax error is corrected.

.-------------------------------------------------------------------------------------------------------------------------------------

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: class is a reserved keyword in Java and cannot be used as an identifier.

Reason:

Reserved keywords in Java are words that have special meanings within the language. They are used to define the structure and syntax of Java programs. Using a reserved keyword as an identifier can lead to ambiguity and confusion for the compiler**.**

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-** When you compile and run this code, you will see the following output:

No parameters

With parameter: 5

Yes, method overloading is allowed in Java. Method overloading means having multiple methods with the same name but different parameters. In this example, we have two display() methods: one takes no parameters and the other takes an integer parameter. The compiler determines which method to call based on the number and types of arguments passed to it.

In the main method, we call display() twice: once without any arguments and once with an integer argument of 5. The compiler correctly identifies and calls the appropriate display() method for each call.

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**- Runtime exception: ArrayIndexOutOfBoundsException

Reason:

This exception occurs because you are trying to access an element in the array arr using an invalid index. The array arr has only three elements (at indices 0, 1, and 2), but you are trying to access the element at index 5, which is beyond the bounds of the array.

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**- When you run this code, it will enter an infinite loop and continue printing "Infinite Loop" indefinitely.

An infinite loop occurs when a loop condition is always true, causing the loop to execute endlessly. In this case, the while loop condition is simply true, which is always true. Therefore, the loop will never terminate.

To avoid infinite loops:

1. Ensure a terminating condition: Make sure that the loop condition eventually becomes false. This can be achieved by:
   * Decrementing or incrementing a counter variable within the loop body.
   * Modifying a condition that determines whether to continue looping.
   * Using a break statement to exit the loop prematurely.
2. Check for edge cases: Consider scenarios that might cause the loop to run indefinitely. For example, if you are using division, make sure you are not dividing by zero.
3. Use a for loop when appropriate: for loops are often used for definite iteration, where the number of iterations is known in advance. This can help prevent accidental infinite loops.

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:

This exception occurs because you are trying to access the length() method of a String object that is null. When a variable is declared with null, it means it doesn't refer to any object. Trying to call a method on a null object results in a NullPointerException.

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-** Compilation error: Incompatible types: String cannot be converted to double.

Reason:

Java is a strongly typed language, which means that every variable must have a specific data type. The data type of a variable determines the range of values it can hold and the operations that can be performed on it.

In the given code snippet, the variable num is declared as a double, which means it can hold floating-point numbers. However, you are trying to assign a String value ("Hello") to num. This is incompatible because String and double are different data types.

Why Java enforces data type constraints:

Java enforces data type constraints to prevent errors and make code more reliable. By ensuring that variables are used correctly, Java can help to avoid common programming mistakes such as:

* Arithmetic errors: If you try to perform arithmetic operations on variables of incompatible types, Java will generate a compilation error.
* Null pointer exceptions: If you try to access a member of a null object, Java will throw a NullPointerException.
* Array index out of bounds exceptions: If you try to access an element of an array using an invalid index, Java will throw an ArrayIndexOutOfBoundsException.
* Type conversion errors: If you try to convert a value of one type to another incompatible type, Java may generate a runtime error or produce unexpected results.

By enforcing data type constraints, Java helps to make code more robust and easier to maintain.

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**- Compilation error: Incompatible types: possible lossy conversion from double to int.

Reason:

This error occurs because you are trying to assign the result of adding an int (num1) and a double (num2) to an int variable (result). The result of adding an int and a double is a double. Assigning a double value to an int variable can lead to loss of precision, as double values can represent a wider range of numbers with higher precision than int values.

How to handle different data types in operations:

1. Cast to the desired data type:
   * If you know that the result of an operation will fit within the range of an int, you can cast the double value to an int before assigning it to the int variable. This will truncate the decimal part of the double value.

Java

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

1. Use the appropriate data type:
   * If you need to preserve the precision of the result, use a double variable to store it.

Java

double result = num1 + num2;

1. Consider using Math.round() or Math.ceil():
   * If you want to round the result to the nearest integer, you can use Math.round().
   * If you want to round the result up to the nearest integer, you can use Math.ceil().

Java

int result = (int) Math.round(num1 + num2);

**-------------------------------------------------------------------------------------------------------------------------------**

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-** The result of this operation is 2.5.

Yes, the output is what you would expect.

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**- The result of this operation is 2.5.

Yes, the output is what you would expect.

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

Operator precedence affects the result as follows:

In Java, the order of operations (operator precedence) is as follows:

1. Parentheses ()
2. Exponentiation ^
3. Multiplication \* and Division /
4. Addition + and Subtraction –

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**- Runtime exception: ArithmeticException

Why division by zero causes an issue in Java:

Division by zero is undefined in mathematics. When you attempt to divide a number by zero, the result is indeterminate. In Java, this leads to an ArithmeticException being thrown.

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- Syntax error: Missing semicolon.

Impact on compilation:

The compiler will encounter this error and fail to compile the code. It will generate an error message indicating that a semicolon is missing at the end of the System.out.println("Hello, World") statement. The compilation process will halt until the syntax error is corrected.

Corrected code:

Java

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!"); // Corrected with semicolon

}

}

With the semicolon in place, the code will compile successfully, and the message "Hello, World!" will be printed to the console when the program is executed.

**------------------------------------------------------------------------------------------------------------------------------------------**

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?

The compiler will generate an error message indicating that there is a mismatch in the curly braces. It will likely say something like:

Error: Error: java: unclosed statement or Error: java: missing closing brace.

This error occurs because the compiler expects a closing brace to terminate the main method block. Without the closing brace, the compiler cannot determine the end of the method and will consider the code after the System.out.println("Hello, World!"); statement as part of the main method. This can lead to further errors or unexpected behavior.

To fix the error, you need to add the missing closing brace at the end of the main method:

Java

public class Main {

public static void main(String[] args) {

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

} // Missing closing brace added here

}

With the closing brace in place, the code will compile successfully and print "Hello, World!" to the console.

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

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?

**Syntax error:** Method cannot be declared within another method.

**Explanation:**

In Java, methods cannot be declared within other methods. This means that the displayMessage() method cannot be declared inside the main method. Methods must be declared at the class level, outside of any other methods.

To fix the error, you need to move the displayMessage() method declaration outside of the main method:

Java

public class Main {

public static void displayMessage() {

System.out.println("Message");

}

public static void main(String[] args) {

displayMessage();

}

}

With the displayMessage() method declared at the class level, you can call it from within the main method.

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

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?

The reason why the default case prints after "Value is 2" in the given switch statement is that there are no break statements between the cases.

In a switch statement, each case label is followed by a block of code. When the value of the expression being switched matches a case label, the corresponding code block is executed. However, if there is no break statement at the end of the code block, the program will continue to execute the code blocks for the following cases until it encounters a break statement or the end of the switch statement.

In this case, the value of value is 2, so the code block for the case 2 label is executed. However, since there is no break statement after the System.out.println("Value is 2"); statement, the program continues to execute the code blocks for case 3 and the default case.

To prevent the program from executing the default case when the value of value is 2, you need to add a break statement at the end of the case 2 code block:

Java

switch (value) {

case 1:

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

break;

case 2:

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

break; // Add break statement here

case 3:

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

break;

default:

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

}

With the break statement in place, the program will stop executing after the case 2 code block, and the default case will not be executed.

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

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?

The reason why the code prints "Level 1", "Level 2", "Level 3", and "Unknown level" when level is 1 is due to the **missing break statements** between the cases.

In a switch statement, each case label is followed by a block of code. When the value of the expression being switched matches a case label, the corresponding code block is executed. However, if there is no break statement at the end of the code block, the program will continue to execute the code blocks for the following cases until it encounters a break statement or the end of the switch statement.

In this case, the value of level is 1, so the code block for the case 1 label is executed. However, since there is no break statement after the System.out.println("Level 1"); statement, the program continues to execute the code blocks for case 2, case 3, and the default case.

**The role of the break statement in this situation is to terminate the execution of the switch statement after the corresponding case block is executed.**

To prevent the program from executing all the cases when level is 1, you need to add a break statement at the end of each case block:

Java

public class MissingBreakCase {

public static void main(String[] args) {

int level = 1;

switch (level) {

case 1:

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

break; // Add break statement here

case 2:

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

break; // Add break statement here

case 3:

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

break; // Add break statement here

default:

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

}

}

}

With the break statements in place, the program will stop executing after the case 1 code block, and the other cases will not be executed. This will result in only "Level 1" being printed when level is 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!");

}

}

}

 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: Incompatible types: possible lossy conversion from double to int.

Explanation:

The switch statement in Java requires the expression being switched to be of an integral type (such as int, byte, short, or char). However, in this code, the score variable is of type double, which is a floating-point type. This mismatch causes the compilation error.

To make the code work, you need to convert the score variable to an integral type before using it in the switch statement. You can do this using a type cast:

Java

public class Switch {

public static void main(String[] args) {

double score = 85.0;

int intValue = (int) score;

switch (intValue) {

case 100:

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

break;

case 85:

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

break;

default:

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

}

}

}

By casting the score variable to an int, you are essentially converting it to its integer equivalent. This allows the switch statement to use the intValue variable as the expression to be switched.

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

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?

Error: Duplicate case label.

Explanation:

In a switch statement, each case label must be unique. Having two identical case labels in the same switch block is not allowed. This is because the compiler cannot determine which code block to execute if the expression being switched matches both case labels.

If you have two identical case labels in a switch block, the compiler will generate an error message indicating that the case label is duplicated. This is to prevent ambiguity and ensure that the switch statement behaves as expected.

To fix the error, you need to either remove one of the duplicate case labels or modify the values of the case labels to make them unique. In the given code, you could remove the second case 5 label or change its value to something different.

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