CDAC Mumbai Lab Assignment : Aditi Mehre_KH

Snippet 1

- **Error:** Main method is not static in class Main, please define the main method as: public static void main(String[] args)
- **Explanation:** The main method should be static.
- **Corrected Code:**

```
public class Main {

public static void main(String[] args) {

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

 } }
```

Snippet 2

- **Error:** Main method is not static in class Main, please define the main method as: public static void main(String[] args)
- **Explanation:** The main method should be public.
- **Corrected Code:**

```
class Main {

public static void main(String[] args) {

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

}

}
```

Snippet 3

- **Error:** Main method must return a value of type void in class Main
- **Explanation:** The main method should return type void. As the main is an entry point of the program, it doesn't require a return value. The Java runtime environment doesn't expect the main method to return any specific value. It simply executes the code within the main method.
- **Corrected Code:**

```
class Main {

 public static void main(String[] args) {

System.out.println("Hello, World!"); return 0; }

 }
```

Snippet 4

- **Error:** Main method not found in class Main
- **Explanation:** The program will compile but not run. String[] args in Java, i.e., a command-line argument array, is used in Java to retrieve input from the console. JVM always looks for the main() method with a string type array as a parameter.
- **Corrected Code:**

```
class Main {

public static void main(String[] args) {

 System.out.println("Hello, World!"); return 0;

}

}
```

Snippet 5

- **Explanation:** You can have multiple main methods with different parameter lists.

Snippet 6

- **Error:** Cannot find the symbol
- **Explanation:** Each variable should be declared because declaring a variable tells the compiler the size and layout and size of memory to be allocated.
- **Corrected Code:**

```
public class Main {

public static void main(String[] args) {

 int y = 2; int x = y + 10; System.out.println(x);

}

 }
```

Snippet 7

- **Error:** String can't be converted into int

- **Explanation:** (No explanation provided in the original prompt)

Snippet 8

- **Error:** Syntax error
- **Explanation:** Syntax is important because it affects the functionality, performance, and quality of your code.
- **Corrected code:**

```
public class Main {

public static void main(String[] args) {

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

 }
```

Snippet 9

- **Errors:**

day2.java:4: error: not a statement int class = 10; ^ day2.java:4: error: ';' expected int class = 10; ^ day2.java:4: error: <identifier> expected int class = 10; ^ day2.java:5: error: illegal start of expression System.out.println(class); ^ day2.java:5: error: <identifier> expected System.out.println(class); ^ 5 errors

- **Explanation:** Keywords can't be used as identifiers because they have reserved meanings. Using them as identifiers will lead to confusion for the compiler.

Snippet 10

- **Errors:**

error: non-static method display() cannot be referenced from a static context display(); ^ day2.java:10: error: non-static method display(int) cannot be referenced from a static context display(5);

- **Explanation:** On running this code, a non-static method can't be called from a static main.
- **Corrected Code:**

```
class Main {
    public static void display() {
        System.out.println("No parameters");
    }

    public static void display(int num) {
        System.out.println("With parameter: " + num);
    }

    public static void main(String[] args) {
        display();
        display(5);
    }
}
```

Snippet 11

- **Error:** Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3 at Main.main(day2.java:4)
- **Explanation:** The 4th index is not present in the given array. The max element the array can have in this program is 3.

Snippet 12

- **Explanation:** while loop with the condition true. This means the loop will keep running forever because the condition is always met. On executing this code, it will continuously print "Infinite Loop" to the console, and it won't stop on its own.

1. **Make sure your loop condition can eventually become false.**
2. **Include a break statement.**

Snippet 13:

```
public class Main {
   public static void main(String[] args) {
          String str = null;
      System.out.println(str.length());
   }
}
```

Error: NullPointerException occurs because we are trying to call a method on a null object reference.

Explanation: NullPointerException is thrown when an App attempts to use null in a case where an object is required.

Corrected Code:

```
public class Main {
   public static void main(String[] args) {
       String str = "Hello"; // Assigning a value to str
      System.out.println(str.length());
   }
   }
```

Snippet 14:

Error: Compilation error because a String cannot be assigned to a double.

Explanation: Java enforces data type constraints, meaning that incompatible data types cannot be directly assigned.

Corrected Code:

```
public class Main {
   public static void main(String[] args) {
         String str = "Hello";
      S ystem.out.println(str);
   }
}
```

Snippet 15:

Error: Compilation error because an int variable cannot hold the result of an addition involving a double.

Explanation: Java needds explicit casting when assigning a value of a larger data type to a smaller one.

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);
   }
}
```

Snippet 16:

```
public class  Main {
```

```
    public static void main(String[] args) {
    int num = 10;
    Double    result = num / 4;
    System.out.println(result);
  }
}
```

Result: 2.0.

Explanation: The division of an int by an int results in an int, so 10 / 4 equals 2. when assigned to a double, it becomes 2.0...

Snippet 17:

Error: Synatx error because the ** operator is not valid in Java.

Explanation: Java does not support ** .

Corrected Code:

```
public class Main {
  public static void main(String[] args) {
  int a =    10;
  int b = 5;
  int result = a * b; System.out .println(result);
   } }
```

Snippet 18:

Output: The output is 20.

Explanation: Operator precedence in Java says that multiplication is performed before addition.

Snippet 19:

Error: ArithmeticException because of division by zero.

Explanation: Division by zero is undefined and causes a runtime exception in Java.

Corrected Code:

```
public class Main {
    public static void main(String[] args) {
          int a = 10;
            int b = 1;
        int result = a / b;
          System.out.println(result);
    }
}
```

Snippet 20:

Error: Syntax error because of the missing semicolon in the end of the statement.

Explanation: A semicolon is required to terminate statements in Java.

Corrected Code:

```
public class Main {
    P ublic static void main(String[] args) {
          System.out.println("Hello, World!");
    }
}
```

Snippet 21:

Error: Syntax error due to missing closing brace.

Explanation: Every openin g brace { must have a closing brace }.

Corrected Code:

```
public class Main {
```

```
    public static void main(String[] args) {


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


  }
}
```

Snippet 22:

Error: Syntax error because a method cannot be declared inside another method.

Explanation: In Java methods should be declared in the class , not inside other methods.

Snippet 23:

Error: The default case is printed after "Value is 2" because there are no break statements.

Explanation: Without break, the program continues executing the next cases even after a match is found.

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");
      }
   }
}
```

Snippet 24:

Error: When level is 1, it prints "Level 1", "Level 2", "Level 3", and "UnknoWn level" because there are no break statements.

Explanation: The break statement Prevents the switch from falling through to other cases.

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");
      }
   }
```

Snippet 25:

Error: The code does not compile because switch cannot handle double types.

Explanation: Java switch statements only accept int, char, byte, short, String, types.

Corrected Code:

```
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:

Error: Duplicate Cases .

Explanation: In a switch , each case must represent a int and unique value. If two identical case are there , the compiler cannot determine which case should be executed when that Value is encounterd.

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;
        case 10: // Changed to a different case value
            System.out.println("This is another case 10");
            break;
     default:
            System.out.println("This is the default case");
    }

}
```

.**Question 1: Grade Classification**

```
import java.util.Scanner;


public class GradeClassification {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the score: ");

        int score = sc.nextInt();


        if (score >= 90) {

            System.out.println("A");

        } else if (score >= 80) {

            System.out.println("B");

        } else if (score >= 70) {

            System.out.println("C");
```

```
        } else if (score >= 60) {
            System.out.println("D");
        } else {
            System.out.println("F");
        }
    }
}
```

Question 2: Days of the Week

```
import java.util.Scanner;

public class DaysOfWeek {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number : ");
        int day = sc.nextInt();

        switch (day) {
            case 1:
                System.out.println("Monday");
```

```
    System.out.println("Weekday");
    break;
case 2:
    System.out.println("Tuesday");
    System.out.println("Weekday");
    break;
case 3:
    System.out.println("Wednesday");
    System.out.println("Weekday");
    break;
case 4:
    System.out.println("Thursday");
    System.out.println("Weekday");
    break;
case 5:
    System.out.println("Friday");
    System.out.println("Weekday");
    break;
case 6:
    System.out.println("Saturday");
    System.out.println("Weekend");
    break;
case 7:
```

```
            System.out.println("Sunday");
            System.out.println("Weekend");
            break;
        default:
            System.out.println("Invalid input");
    }
  }
}
```

Question 3: Calculator

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
```

```
System.out.print("Enter an operator (+, -, *, /) ");
char operator = sc.next().charAt(0);

double result;

switch (operator) {
    case '+':
        result = num1 + num2;
        System.out.println("Result: " + result);
        break;
    case '-':
        result = num1 - num2;
        System.out.println("Result: " + result);
        break;
    case '*':
        result = num1 * num2;
        System.out.println("Result: " + result);
        break;
    case '/':
        if (num2 == 0) {
            System.out.println("Error: Division by zero");
        } else {
            result = num1 / num2;
```

```
                System.out.println("Result: " + result);
            }
            break;
        default:
            System.out.println("Invalid operator");
    }
  }
}
```

Question 4: Discount Calculation

```
import java.util.Scanner;

public class DiscountCalculation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the total purchase amount: ");
        double totalPurchase = sc.nextDouble();
        System.out.print("Do you have a membership card? (yes/no): ");
        String hasMembership = sc.next();
```

```
        double discount = 0;

        if (totalPurchase >= 1000) {
            discount = 0.20;
        } else if (totalPurchase >= 500) {
            discount = 0.10;
        } else {
            discount = 0.05;
        }

        if (hasMembership.equals("yes")) {
            discount += 0.05;
        }

        double finalAmount = totalPurchase - (totalPurchase * discount);
        System.out.println("Final amount after discount: Rs." + finalAmount);
    }
}
```

1. Evolution of Programming Languages

Low-Level Language: These are closer to machine language, like assembly language. They give you control over hardware but are harder to write and understand. Example: Assembly language.

High-Level Language: These are easier to read and write because they are closer to human language. They hide complex details from the programmer. Example: Python, Java.

Assembly-Level Language: This is a type of low-level language that uses short codes (like MOV for move) to perform tasks. It’s a bit easier than machine code but still close to how the computer's hardware operates.

Why Different Levels? We need different levels because sometimes we want easy programming (high-level),

and other times we need more control over the hardware (low-level). High-level is simpler but less powerful in controlling hardware, while low-level is harder but gives more control.

2. Different Programming Languages and Their Usage

C: Powerful for system programming (like operating systems), but tricky with memory management.

Python: Easy to learn and great for data science and scripting, but slower in execution.

Java: Good for building cross-platform applications, but can be slower to start up.

JavaScript: Essential for web development, but only works well in browsers.

C++: Offers both low-level and high-level features, but complex syntax.

Ruby: Easy and fun for web development, but slower and less popular than others.

Go: Fast and great for building web servers, but still growing in popularity.

Choosing a Language: For web development, pick JavaScript because it works directly in browsers. For data science, Python is popular because it has many libraries for data analysis.

One Language for Everything? Each language has strengths for certain tasks. For example, Python is great for data science, but not ideal for building a game engine, where C++ might be better.

3. Which Programming Language is the Best?

Best Language? There's no one "best" language. It depends on what you're trying to do. Python is great for beginners, but C is better for systems programming. Why Not Use One Language Everywhere? Different tasks need different tools. Companies pick languages based on cost, performance, and what their developers know.

Trends in Languages: New languages like Rust are getting popular because they offer safety and performance,

which older languages struggle with.

4. Features of Java

Platform-Independent: Java programs run on any device with a Java Virtual Machine (JVM). This is why it's considered platform-independent.

Robust: Java is robust because it handles memory management and exceptions well, preventing many errors.

Secure: Java is secure because it checks code for harmful behavior (bytecode verification) and manages memory automatically (garbage collection).

Other Features: Java's multithreading allows multiple tasks to run at once. Its portability and simplicity make it easy to use

across different platforms.

5. Role of `public static void main(String[] args)` (PSVM)

public: Makes the method accessible from anywhere.

static: Means the method belongs to the class, not instances.

void: The method doesn’t return any value.

main: It’s the entry point of the program where the execution starts.

Removing Keywords: If you remove or change these keywords, the program won’t run correctly, and you'll get errors.

String[] args: This allows the program to accept input

from the command line. If you leave it out, the program won’t accept those inputs.

6. Can We Write Multiple `main` Methods?

Multiple `main` Methods in a Class? You can’t have more than one main method in the same class with the same signature. It will cause a compilation error.

Multiple `main` Methods in Different Classes? Yes, you can have main methods in different classes. The JVM will start with the one in the class you specify.

Overloading `main` Method:

You can overload it with different parameters, but the JVM will always look for main(String[] args) to start the program.

7. Naming Conventions in Java

Uppercase vs. Lowercase: Class names start with an uppercase letter (like MyClass), while variables and method names start with a lowercase letter (like myVariable). This helps in identifying them quickly.

Rules for Naming: Variables and methods should start with a letter, and names should be descriptive but short. Following these rules makes the code easier to read and maintain, especially in big projects.

8. Java Object Creation and Memory Management

Objects on the Heap: Java creates objects on the heap because it's a larger memory area where objects can grow dynamically. This allows Java to manage memory efficiently.

Memor y Management: Java uses garbage collection to automatically free up memory by deleting objects that are no longer in use, so you don't have to do it manually.

Overloading vs. Overriding:

Overloading allows methods to have the same name but different parameters, while overriding means a subclass provides a specific implementation of a method from its superclass.

Classes and Objects: Java uses classes to define blueprints for objects, which are instances that follow the principles of OOP like encapsulation, inheritance, and polymorphism.

9. Purpose of Access Modifiers in Java

Purpose of Access Modifiers: T hey control

who can access a class, method, or variable. For example, public means anyone can access it, while private restricts it to the class itself.

Contribution to OOP: Access modifiers help protect data and enforce encapsulation, ensuring that only the necessary parts of the code can interact with specific variables or methods.