Sure! Here's a **line-by-line explanation** of your corrected code:

**✅ Class Definition**

public class prac3 {

* This line defines a **class** named prac3.
* public means this class is accessible from outside its package or file.

public Double myValue = 0.0;

* Declares a **public variable** named myValue of type Double (a decimal number).
* It is initialized to 0.0.

public Double getUpdatedValue() {

* This line defines a **method** named getUpdatedValue.
* public means it can be called from outside the class.
* It returns a value of type Double.

myValue = myValue + 10;

* This updates myValue by adding 10 to its current value.
* Since myValue started at 0.0, it becomes 10.0 after this line.

return myValue;

}

* The method **returns** the updated value of myValue.

**✅ Main Logic to Use the Class**

public class TestClass {

* Defines another class named TestClass, which contains the main method to run the program.

public static void main(String[] args) {

* This is the **entry point** of the program.
* The Java Virtual Machine (JVM) starts execution from this method.

prac3 m = new prac3();

* Creates an object m of the class prac3.
* This allows us to access its methods and variables.

Double myValue = m.getUpdatedValue();

* Calls the method getUpdatedValue() on object m.
* It updates the internal variable myValue to 10.0 and returns that value.
* The result is stored in a new local variable also named myValue.

System.out.println("valued = " + myValue);

* This line **prints** the output to the console.
* It will show: valued = 10.0

Let me know if you'd like to make this dynamic (e.g., accepting a custom increment or input from the user).

**✅ 1. What does your Apex application do?**

**Answer:**  
The application defines a class with a method that increases a numeric value by 10 and returns the updated value. It demonstrates basic arithmetic and method calling in Apex.

**✅ 2. Can you show the code for the incremental function?**

**Answer:**

apex

CopyEdit

public class Incrementer {

public Integer myValue = 0;

public Integer increaseByTen() {

myValue = myValue + 10;

return myValue;

}

}

**✅ 3. How do you call this method in Salesforce?**

**Answer:**  
Using the **Developer Console's Anonymous Window**:

apex

CopyEdit

Incrementer obj = new Incrementer();

Integer updatedValue = obj.increaseByTen();

System.debug('Updated Value = ' + updatedValue);

**✅ 4. What is the purpose of myValue in the class?**

**Answer:**  
myValue is an instance variable that holds the current value. The method increaseByTen() updates it by adding 10 each time it’s called.

**✅ 5. What is the default value of an Integer in Apex?**

**Answer:**  
The default value of an uninitialized Integer in Apex is null, but here it is explicitly initialized to 0.

**✅ 6. Can this function be modified to accept a custom increment value?**

**Answer:**  
Yes, we can pass the increment as a parameter:

apex

CopyEdit

public Integer increaseBy(Integer increment) {

myValue += increment;

return myValue;

}

**✅ 7. How would you write a test class for this?**

**Answer:**

apex

CopyEdit

@isTest

private class IncrementerTest {

@isTest static void testIncreaseByTen() {

Incrementer inc = new Incrementer();

Integer result = inc.increaseByTen();

System.assertEquals(10, result);

}

}

**✅ 8. Why is System.debug() used?**

**Answer:**  
System.debug() prints output to the **Debug Logs**, which helps developers view runtime results and diagnose issues during testing.

**✅ 9. Can this class be made more reusable?**

**Answer:**  
Yes, by making the method static or allowing user input to set the starting value or increment.

**✅ 10. How does this program help you understand Apex basics?**

**Answer:**  
It helps me understand variable scope, method definition, object instantiation, and using Developer Console to execute and test Apex code.

Calculator

-----

Absolutely! Let's go through your Apex **calculator code line by line** and explain each part clearly:

**🔢 Your Code:**

public class Calculator {

public static Integer calculate(Integer a, Integer b, String op) {

if (op == '+') {

return a + b;

} else if (op == '-') {

return a - b;

} else if (op == '\*') {

return a \* b;

} else if (op == '/') {

return a / b;

} else {

throw new IllegalArgumentException('Invalid operator');

}

}

}

Integer n1 = 3;

Integer n2 = 5;

System.debug('Addition: ' + Calculator.calculate(n1, n2, '\*'));

**✅ Line-by-Line Explanation**

**public class Calculator {**

* This line defines a **class** named Calculator.
* public means it can be accessed from other classes.

**public static Integer calculate(Integer a, Integer b, String op) {**

* Declares a **static method** named calculate.
* It takes two numbers (a and b) and a **String** op (which represents the operator like +, -, \*, /).
* It returns an Integer result.

**if (op == '+') { return a + b; }**

* Checks if the operator is '+'. If yes, returns the **sum** of a and b.
* 🔴 *Note:* This uses == to compare strings, which is **not recommended in Apex**. You should use op.equals('+') instead.

**else if (op == '-') { return a - b; }**

* If the operator is '-', returns the **difference** between a and b.

**else if (op == '\*') { return a \* b; }**

* If the operator is '\*', returns the **product** of a and b.

**else if (op == '/') { return a / b; }**

* If the operator is '/', returns the **quotient** of a divided by b.
* ⚠️ No check for **division by zero**, which can cause a runtime error.

**else { throw new IllegalArgumentException('Invalid operator'); }**

* If the operator is none of the above, throws an **exception** to indicate an invalid input.

**👇 Calling Code**

**Integer n1 = 3;**

* Declares an integer variable n1 with value 3.

**Integer n2 = 5;**

* Declares an integer variable n2 with value 5.

**System.debug('Addition: ' + Calculator.calculate(n1, n2, '\*'));**

* Calls the calculate method with 3, 5, and '\*'.
* Prints: Addition: 15
* ❗ This line labels the output as “Addition” even though the operation is multiplication. Consider renaming it to:
* System.debug('Multiplication: ' + Calculator.calculate(n1, n2, '\*'));

**🛠 Suggestions for Improvement**

1. **Use equals() for string comparison:**
2. if (op.equals('+')) { ... }
3. **Add a division-by-zero check:**
4. if (b == 0) throw new ArithmeticException('Cannot divide by zero');

Would you like me to provide a test class for this calculator too?

Thanks! Based on your **practical viva problem statement** — *"Create an application in Salesforce using Apex to implement a simple calculator"* — here are **10 relevant viva questions and answers** tailored specifically to this task.

**🎓 Apex Calculator Application – Viva Q&A**

**✅ 1. What is Apex, and why is it used in Salesforce?**

**Answer:**  
Apex is a strongly-typed, object-oriented programming language developed by Salesforce. It is used to write custom business logic like triggers, classes, and integrations within the Salesforce platform.

**✅ 2. How did you implement the calculator in Apex?**

**Answer:**  
I created an Apex class named Calculator with methods like add, subtract, multiply, and divide, each accepting two parameters and returning a result. I then used a test class or anonymous window to execute and test the operations.

**✅ 3. Can you show a sample method from your Calculator class?**

**Answer:**

public class Calculator {

public static Decimal add(Decimal a, Decimal b) {

return a + b;

}

}

**✅ 4. How do you call this method in Salesforce?**

**Answer:**  
I can call it using **Anonymous Apex Execution** in the Developer Console:

Decimal result = Calculator.add(10, 5);

System.debug('Addition = ' + result);

**✅ 5. How do you handle division by zero in your calculator?**

**Answer:**  
I added a condition to check if the denominator is zero. If it is, the method throws a custom error message.

public static Decimal divide(Decimal a, Decimal b) {

if (b == 0) {

throw new ArithmeticException('Cannot divide by zero');

}

return a / b;

}

**✅ 6. What data type did you use and why?**

**Answer:**  
I used the Decimal data type because it supports floating point arithmetic and is suitable for precise financial or mathematical calculations.

**✅ 7. How would you expose this calculator to a Visualforce Page or Lightning Component?**

**Answer:**  
I would annotate the class with @AuraEnabled (for LWC) or remoteAction (for Visualforce) so that frontend components can call the methods in the Apex class.

**✅ 8. What is the difference between a static and non-static method in Apex, and what did you use?**

**Answer:**

* **Static** methods can be called without creating an object.
* I used **static methods** because they’re simple utility functions and don’t need to maintain state.

**✅ 9. Did you write a test class for your calculator?**

**Answer:**  
Yes. I wrote a test class using @isTest to cover all operations and validate the correctness of the methods.

Example:

@isTest

private class CalculatorTest {

@isTest static void testAdd() {

System.assertEquals(15, Calculator.add(10, 5));

}

}

**✅ 10. How can this calculator application be improved or extended?**

**Answer:**  
It can be extended by:

* Adding more operations like modulus, power, square root
* Creating a Visualforce or LWC UI
* Logging operations in a custom object for history tracking

Would you like me to help you create the **full Apex calculator code** and test class too?