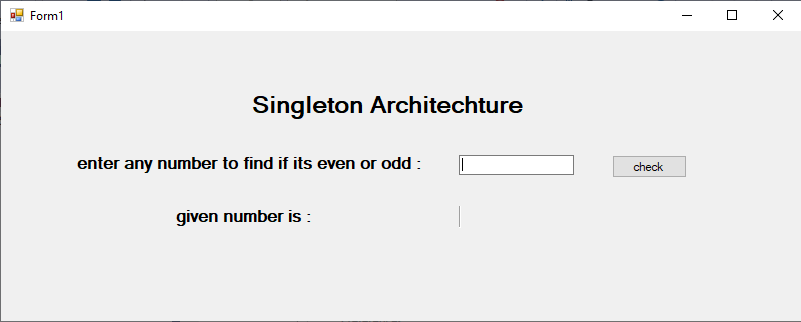
**Creational Pattern**

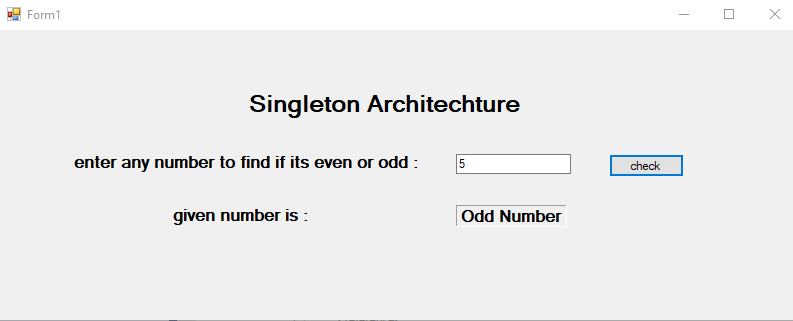
(singleton pattern)

First, we have made a program that will tell us whether the given number in even or odd. Its architecture is based on Singleton method, which is a type of Creational Pattern.

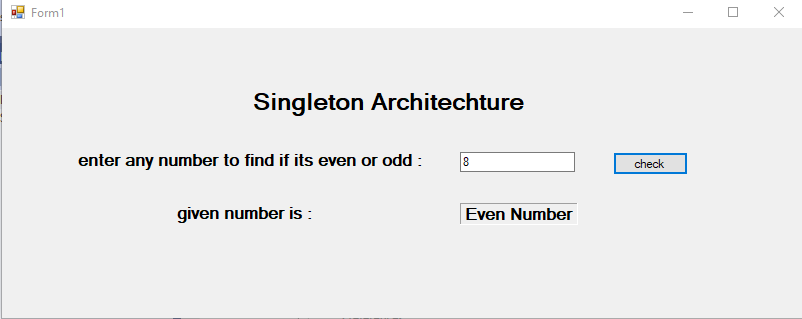
Output of code :



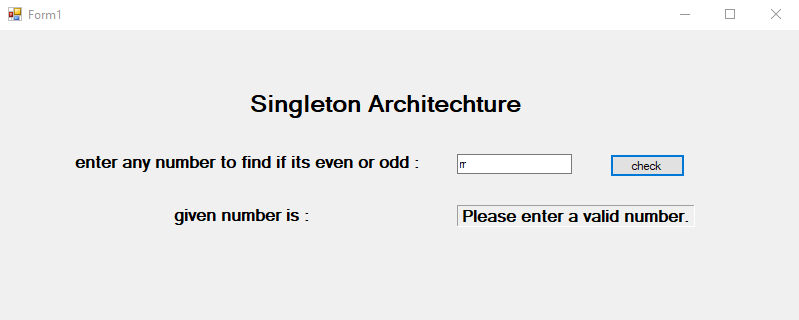
At first, it will ask for a number from user



After entering the number, it will display “odd number” if its an odd number



It will display “even number” if its an even number

it will display “Please enter a valid number” if its something other than integer value

**Working and Anatomy of code**

### Flow of Execution:

1. **btnCheck\_Click**:
   * This method gets triggered when the user clicks the button (btnCheck).
   * The btnCheck\_Click method is linked to the button's Click event. When the user clicks the button, the event handler (btnCheck\_Click) executes.
2. **Get Singleton Instance**:
   * EvenOddChecker checker = EvenOddChecker.GetInstance();
     + Here, we are calling the GetInstance method of the EvenOddChecker class.
     + This ensures that we get a single instance of the EvenOddChecker class (which is the core idea of the Singleton pattern).
     + The GetInstance method checks whether an instance of EvenOddChecker already exists. If not, it creates one. This guarantees that only one instance is used throughout the program.
3. **Parsing User Input**:
   * if (int.TryParse(txtNumber.Text, out int number)) { ... }
     + The program attempts to parse the text entered by the user (txtNumber.Text) into an integer using int.TryParse().
     + TryParse is a safe way to convert the string input into an integer, returning true if the conversion is successful and false otherwise.
     + If the conversion is successful, the program proceeds to check if the number is even or odd.
     + If the user input is invalid (not a number), the program goes to the else block, where a message prompts the user to enter a valid number.
4. **Checking If the Number is Even or Odd**:
   * string result = checker.CheckEvenOdd(number);
     + If the number is successfully parsed, the CheckEvenOdd method of the EvenOddChecker class is called to check whether the number is even or odd.
     + This method takes an integer (number) and checks the condition number % 2 == 0:
       - If the number is divisible by 2 (i.e., the remainder is 0), the number is even, and the method returns "Even Number".
       - Otherwise, it returns "Odd Number".
5. **Displaying the Result**:
   * lblResult.Text = result;
     + Once the result is determined (either "Even Number" or "Odd Number"), the result is displayed on the label (lblResult.Text).
     + If the user input is invalid, the message "Please enter a valid number." is displayed on the label.
6. **Else Block** (for invalid input):
   * If the input is not a valid number (int.TryParse returns false), the program will skip the CheckEvenOdd method and display "Please enter a valid number." on the label.

### Anatomy of the Code:

#### 1. btnCheck\_Click Event Handler:

This is the event handler method linked to the Click event of the button. It interacts with the user interface components like the text box (txtNumber) and label (lblResult) and controls the flow of the logic based on user input.

#### 2. Singleton Pattern (EvenOddChecker Class):

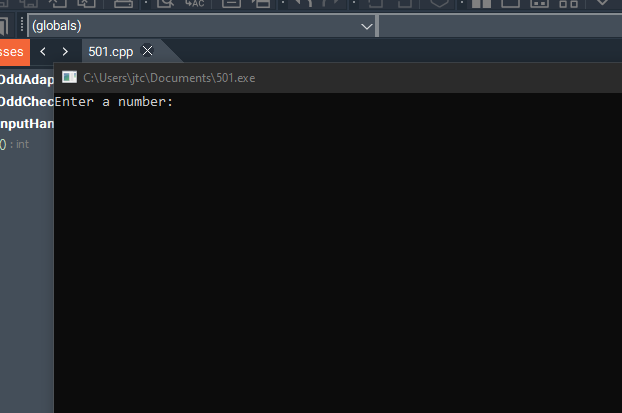
* **Private Static Variable (\_instance)**:
  + This is a static variable that holds the single instance of the EvenOddChecker class. It's initially null.
* **Private Constructor**:
  + The constructor is private to prevent direct instantiation from outside the class. This ensures that no other instance of EvenOddChecker can be created manually.
* **Public Static Method (GetInstance)**:
  + This is the method that provides access to the single instance of the class. If \_instance is null, it creates a new EvenOddChecker object. If it already exists, it simply returns the existing instance. This is the heart of the Singleton pattern.
* **Method to Check Even or Odd (CheckEvenOdd)**:
  + This method performs the actual logic of checking whether a number is even or odd by using the modulus operator (%). If number % 2 == 0, the number is even; otherwise, it's odd.

**Structural Pattern**

(Adapter pattern)

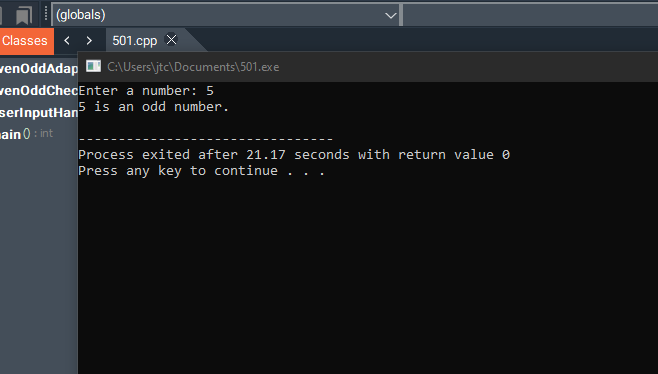
Secondly, we have made a program that will tell us whether the given number in even or odd. Its architecture is based on Adaptor method, which is a type of Structural Pattern.

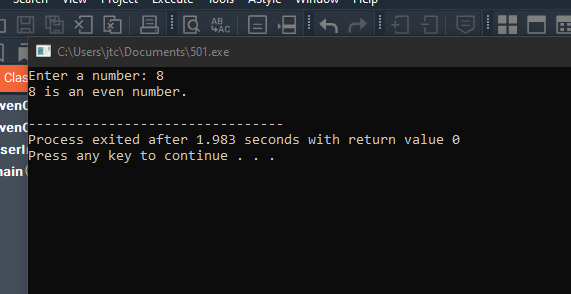
Output of the code :



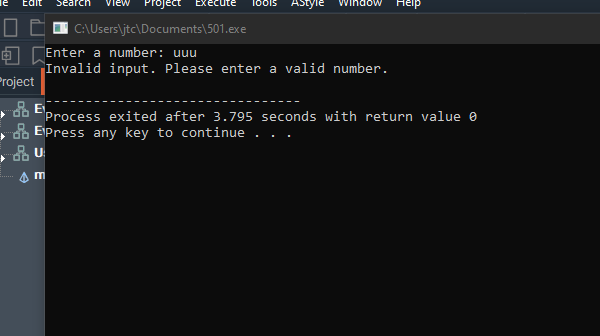
At first, it will ask for a number from user

After entering the number, it will display “odd number” if its an odd number



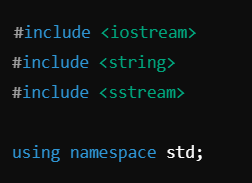


It will display “even number” if its an even number

It will display “Please enter a valid number” if its something other than integer value

### ****Anatomy of the Code****

#### 1. **Headers and Namespace**



1 #include <iostream>: Enables input (cin) and output (cout) operations.

2 #include <string>: Allows the program to handle strings.

3 #include <sstream>: Provides tools like stringstream for converting strings to other data types (used for input validation).

4 using namespace std;: Lets us use standard library features like cin and cout without prefixing them with std::

#### 2. **Class: UserInputHandler**

This class handles the user input and validates it.

##### Key Method:

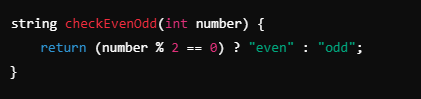
**How It Works**:

1. **Takes input as a string**:
   * The user types something, and it is stored in a string variable (input).
2. **Uses stringstream to attempt conversion**:
   * A stringstream object (ss) is used to try converting the string to an integer. This helps us handle cases where the input is invalid (like letters or mixed characters).
3. **Validates the conversion**:
   * ss.fail(): Checks if the conversion failed (e.g., input is "abc").
   * !ss.eof(): Ensures there are no leftover characters after a valid number (e.g., input is "12abc").
   * If either condition is true, the method returns false to indicate invalid input.
4. **Returns true for valid input**:
   * If the input is successfully converted to an integer and there are no extra characters, the method returns true and stores the number in the number reference variable.

#### 3. **Class: EvenOddChecker**

This class contains the logic to determine whether a number is even or odd.

##### Key Method:



**How It Works**:

* The method uses the modulo operator (%) to check if the number is divisible by 2.
* If the remainder is 0, the number is even; otherwise, it’s odd.
* Returns "even" or "odd" as a string result

#### 4. **Class: EvenOddAdapter**

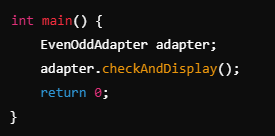
This is the **adapter** that connects UserInputHandler and EvenOddChecker. It manages the overall flow of logic.

##### Key Method:

**How It Works**:

1. **Gets input**:
   * Calls the getNumber method of UserInputHandler to read and validate the user’s input.
2. **Handles valid input**:
   * If getNumber returns true, the program passes the valid number to the checkEvenOdd method of EvenOddChecker.
   * Displays the result (e.g., "42 is an even number").
3. **Handles invalid input**:
   * If getNumber returns false, it displays an error message: "Invalid input. Please enter a valid number."

#### **Main Function:**



**How It Works**:

1. **Creates an EvenOddAdapter object**:
   * This object is the main point of interaction between input handling and even-odd checking.
2. **Calls the checkAndDisplay method**:
   * Manages the entire workflow: input validation, checking even/odd, and displaying results

### ****Detailed Flow of Execution****

#### **Input: 42**

1. **Main Function**:
   * EvenOddAdapter adapter;
   * adapter.checkAndDisplay();
2. **Adapter Calls Input Handler**:
   * inputHandler.getNumber(number) asks the user to enter a number.
   * User enters "42". The string is converted to 42 and validated successfully.
3. **Adapter Calls Checker**:
   * checker.checkEvenOdd(42) checks if the number is even or odd.
   * Returns "even".
4. **Result is Displayed**:
   * 42 is an even number.

#### **Input: abc**

1. **Main Function**:
   * EvenOddAdapter adapter;
   * adapter.checkAndDisplay();
2. **Adapter Calls Input Handler**:
   * inputHandler.getNumber(number) asks the user to enter a number.
   * User enters "abc". Validation fails because "abc" cannot be converted to an integer.
3. **Adapter Handles Invalid Input**:
   * Displays: Invalid input. Please enter a valid number.

### ****Key Features and Concepts****

1. **Validation**:
   * The program ensures only valid integers are processed, preventing incorrect behavior.
2. **Separation of Concerns**:
   * Input handling, number checking, and program control are split into three separate classes, improving readability and maintainability.
3. **Adapter Pattern**:
   * The EvenOddAdapter serves as the bridge between input validation (UserInputHandler) and even/odd logic (EvenOddChecker).
4. **Error Handling**:
   * Proper error messages guide the user when invalid input is entered.

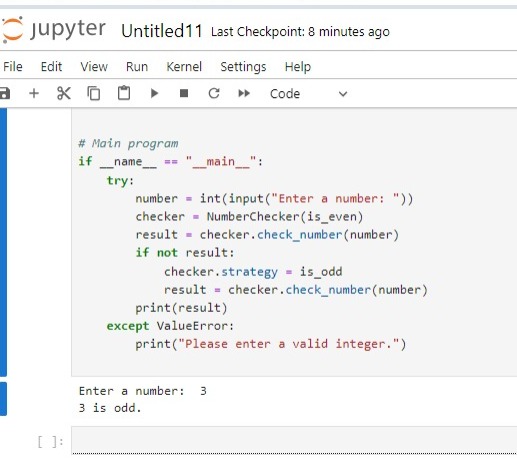
**Behavioral Pattern**

(strategy pattern)

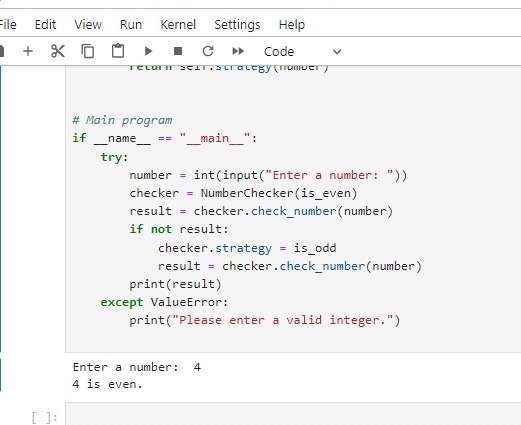
First, we have made a program that will tell us whether the given number in even or odd. Its architecture is based on Observer method, which is a type of Creational Pattern.

Output of code :

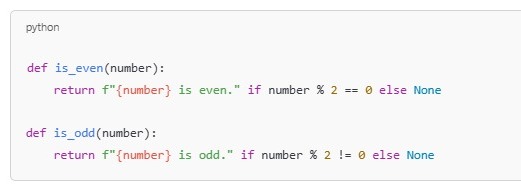
At first, it will ask for a number from user

After entering the number, it will display “odd number” if its an odd number

It will display “even number” if its an even number



### ****Anatomy of the Code****



Strategy Functions

How it works

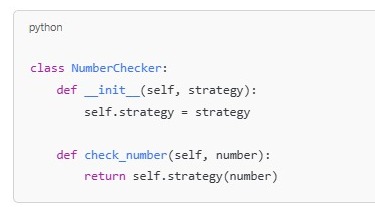
These are the "strategies" used for checking whether a number is even or odd.

They encapsulate the logic for determining the number's nature.

is\_even : Checks if the number is divisible by 2 without a remainder (number % 2 == 0).

is\_odd : Checks if the number has a remainder when divided by 2 (number % 2 != 0).

Both return a formatted string if the condition is true or None if false.

  
Context Class

How it works

The NumberChecker class represents the context in the Strategy Pattern.

It uses a strategy to perform the desired operation.

\_init\_ : Accepts a strategy function (like is\_even or is\_odd) and assigns it to the strategy attribute.

check\_number : Executes the strategy function (self.strategy) with the given number.

The context is reusable and allows dynamic switching of strategies.

Main Program

The main program interacts with the user, uses the context, and switches strategies if needed.

Input Handling : Accepts input and converts it to an integer.

First Strategy : Starts with the is\_even strategy.

Strategy Switch : If is\_even fails (returns None), switches to the is\_odd strategy.

Error Handling : If the input isn't a valid integer, an error message is displayed.

**Flow of Execution**

Input: User enters a number (e.g., 7).

Initial Strategy: The NumberChecker is initialized with is\_even.

It checks if the number is even.

Strategy Switch (if needed):

If the first strategy fails, it switches to is\_odd and checks again.

Output: Prints whether the number is even or odd