Part01:

Github Repo Link (contains all problems):

https://github.com/janaashraf888/rowadmasrtasks.git

Difference between int.Parse and Convert.ToInt32 when handling null inputs:

When a null value is passed as input, int.Parse and Convert.ToInt32 behave differently.
int.Parse throws an ArgumentNullException because it expects a valid numeric string.
On the other hand, Convert.ToInt32 does not throw an exception when the input is null; instead, it returns the value 0.

Why TryParse is recommended over Parse in user-facing applications:

TryParse is recommended because it **safely checks the user input without causing errors or crashing the program**.
Unlike Parse, which throws an exception if the input is invalid, TryParse returns false when the input cannot be converted. This allows the program to handle the error gracefully, display a friendly message to the user, and continue running.

Real Purpose of GetHashCode():

The real purpose of the GetHashCode() method is to provide a **numeric value (hash code)** that represents an object. This hash code is used mainly for **efficient data lookup and storage**, especially in collections such as **Dictionary**, **HashSet**, and **Hashtable**.

A hash code helps the collection quickly locate the object by using the hash code as an index instead of searching through all elements. It is not meant to uniquely identify an object, because different objects can have the same hash code (called a **hash collision**).

Significance of Reference Equality in .NET:

Reference equality in .NET means that **two variables refer to the exact same object in memory**. It is significant because it helps determine whether two references point to the same instance, not just whether they contain equal values.

Reference equality is important when working with **reference types** (such as classes, arrays, and strings). It is used to compare objects in memory, especially in cases where object identity matters, such as:

- Checking whether two references point to the same instance
- Preventing duplicate objects in collections
- Managing shared resources and memory optimization
- Avoiding unexpected changes when multiple references modify the same object

Why String Is Immutable in C#:

Strings in C# are immutable because it improves **security, performance, and thread safety**.
Immutable strings can be safely shared and reused in memory, and their hash codes never change, which is important when using strings as keys in collections like Dictionary.

How StringBuilder Fixes String Concatenation Inefficiency:

In C#, strings are **immutable**, meaning each time you concatenate strings using the + operator, a **new string object is created**. This uses more **memory** and slows performance, especially when concatenating many strings inside loops.

StringBuilder is **mutable**, so it modifies the existing string buffer instead of creating a new object each time. This makes string operations **faster and more memory-efficient**, especially when combining many strings.

Why StringBuilder Is Faster for Large-Scale String Modifications:

StringBuilder is faster because it is **mutable**, meaning it modifies the same memory buffer instead of creating a new string each time.
When many changes (like appending, inserting, or removing) are made, StringBuilder avoids repeated memory allocation and copying, which makes it much more efficient than using string concatenation.

Most Used String Formatting Method in C# and Why:

The most used string formatting method in C# is **string interpolation** ($"...").

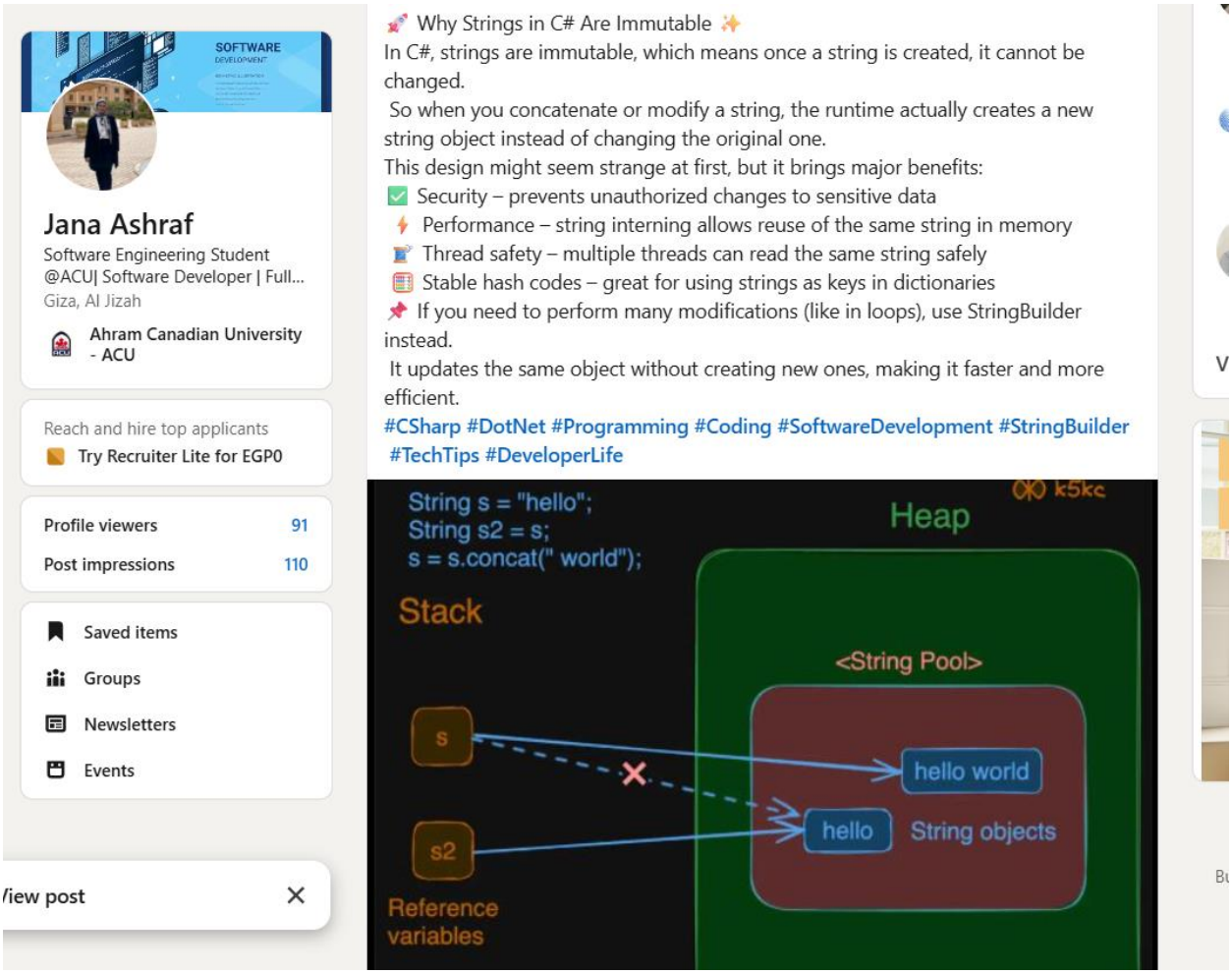
It is the most popular because it is **simple, readable, and easy to write**, especially when inserting multiple variables into a string. It makes the code cleaner compared to concatenation or string.Format, and it reduces mistakes while improving readability.

How StringBuilder Handles Frequent Modifications Compared to Strings:

StringBuilder is designed for frequent string changes because it is **mutable**, meaning its content can be modified directly without creating a new object each time. It uses a **dynamic internal buffer** that can grow as needed. When you append, insert, replace, or remove text, StringBuilder updates this buffer in place.

In contrast, strings are **immutable**, so any modification creates a new string object and copies the old value into it. This makes repeated modifications slow and memory-intensive. Therefore, StringBuilder is more efficient for operations that involve many changes.

Part02:



What is an Enum Data Type?

An Enum (enumeration) is a special data type in C# that allows you to define a group of named constant values. Enums make your code easier to read and maintain by replacing numeric values with meaningful names.

When is it used?

Enums are used when you have a fixed set of related values that represent a category or choice, such as:

- Days of the week
- Months of the year
- Status values (e.g., Active, Inactive, Pending)
- Options in menus or settings

Enums help avoid using magic numbers and make the code more understandable.

Three Common Built-in Enums Used Frequently

1. **ConsoleColor**
Used to set the color of text or background in the console.

2. **DayOfWeek**
Represents the days of the week.

3. **FileMode**
Used to specify how a file should be opened or created (e.g., Open, Create, Append).

When to Use string vs StringBuilder

Use string when:

- You have **small or simple text** operations.
- You perform **few or no modifications**.
- You need **readable and clean code**.
- You are working with **fixed text values**.

Example:

```
string name = "Willy";

string greeting = name + " Hi!";
```

Use StringBuilder when:

- You need **frequent or large modifications** (e.g., inside loops).
- You concatenate or modify strings many times.
- Performance and memory efficiency matter.
- You are building a large text result.

Example:

```
StringBuilder sb = new StringBuilder();

for (int i = 0; i < 1000; i++)

{

sb.Append(i);

}
```

Part03 Bonus:

What is a User-Defined Constructor?

A **user-defined constructor** is a constructor that you **write yourself** inside a class.
It is used to initialize the object when it is created.

A constructor has the **same name as the class** and does not have a return type.

Role in Initialization

The main role of a constructor is to **set initial values** for the object's properties or fields.
When you create an object using new, the constructor runs automatically and prepares the object for use.

Example:

```
class Student
{
public string Name;
public int Age;

// User-defined constructor
public Student(string name, int age)
{
Name = name;
Age = age;
}
}
```

Array vs Linked List:

Array:

- Fixed size
- Fast random access using index (O(1))
- Slow insertion/deletion (O(n))

Linked List:

- Dynamic size
- Slow random access (O(n))
- Fast insertion/deletion if node reference is known (O(1))