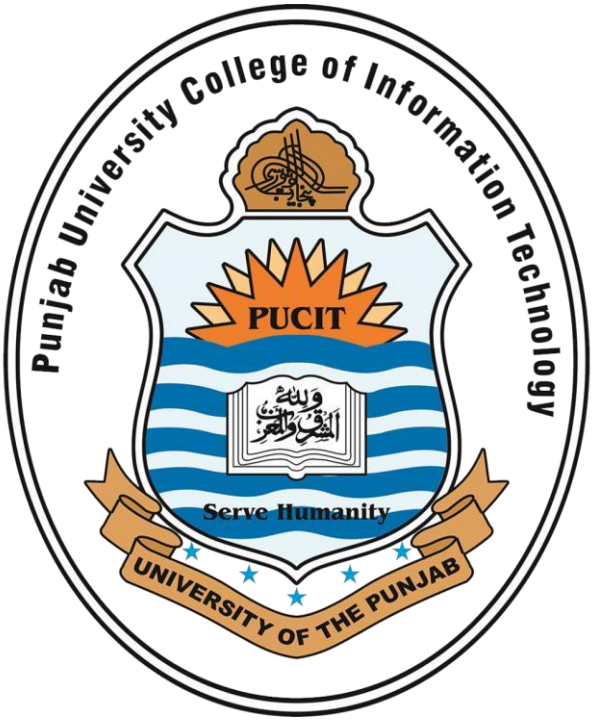
PUNJAB UNIVERSITY COLLEGE OF INFORMATION AND TECHNOLOGY



Submitted to: Ma’am Fatima Sabir

Submitted by: MUHAMMAD HANNAN KHALID
Roll no: BCSF20A047

ASSIGNMENT 2

Enterprise Application Development

(EAD)

PUNJAB UNIVERSITY COLLEGE OF INFORMATION TECHNOLOGY

Q1

Differentiate between Generic vs non -Generic Collection in C#.

Generic Collections:

Generic collections in C# are designed to work with strongly-typed data. They allow you to store and manipulate collections of objects of a specific data type. Here are some key points about generic collections:

- **Strongly Typed**: Generic collections are strongly typed, which means you specify the data type they will contain when you declare them. For example, you can have a List<int> to store integers or a List<string> to store strings.

- **Type Safety**: Because of their strong typing, generic collections provide type safety. This means that you can only add and retrieve elements of the specified data type, reducing the risk of runtime errors.

- **Compile-Time Checks**: The data type is checked at compile time, so if you try to use the wrong data type, you'll get a compilation error.

Examples: Some common generic collections in C# include List<T>, Dictionary<TKey, TValue>, and Queue<T>.

Non-Generic Collections:

Non-generic collections, on the other hand, are not strongly typed. They can store objects of any data type, making them less type-safe. Here are some key points about non-generic collections:

- **Not Strongly Typed**: Non-generic collections can store objects of any type, as they are not bound to a specific data type.

- **Lack of Type Safety**: Because non-generic collections can hold any type of object, they don't provide type safety. You can encounter runtime errors if you try to retrieve an object of the wrong type.

Examples: Some common non-generic collections include ArrayList and Hashtable.

Q2

What collection is suitable for Form Design entry based on the user needs vs Form design entries that are fixed?

Form Design Entry Based on User Needs:

When a user is dealing with form entries that can vary in data types based on user input, it's best to use a dynamic data structure. In C#, you can use a List<object> or an ArrayList. Here's why:

- **Flexibility :** These collections can store objects of various data types, making them suitable for accommodating different user inputs like text, numbers, dates, etc.
- **User-Friendly :** They allow you to capture user input without strict data type restrictions, providing flexibility to adapt to different user needs.

Form Design Entry Based on User Needs:

When you're dealing with form entries that can vary in data types based on user input, it's best to use a dynamic data structure. In C#, you can use a List<object> or an ArrayList. Here's why:

- **Flexibility:** These collections can store objects of various data types, making them suitable for accommodating different user inputs like text, numbers, dates, etc.
- **User-Friendly:** They allow you to capture user input without strict data type restrictions, providing flexibility to adapt to different user needs.

Conclusion:

- When your form entries can vary in data types based on user input, use a dynamic collection like List<object> or ArrayList for flexibility.
- For fixed form design entries with known data types, choose a strongly-typed collection like List<string> or List<int> for better type safety and code clarity.