**Enterprise Application Development**

**Assignment**

**Name: Muhammad Umar**

**Roll no: BCSF20A012**

**Ans no 1:**

**Non-Generic Collections:**

Non-generic collections, such as ArrayList, Hashtable, and SortedList, can store objects of any data type because they store elements as objects (type **object**). When you retrieve objects from these collections, you need to cast them back to their original data types, which can lead to runtime errors if not done correctly.

**Advantages of Non-Generic Collections:**

* **Flexibility:** Non-generic collections can store heterogeneous data types in the same collection.
* **Simplicity:** They are easier to use when you don't need type safety or when dealing with different types of objects.
* **Flexibility**: Non-generic collections can store heterogeneous data types because they treat everything as an object.
* **Type Casting:** When retrieving elements, you need to explicitly cast them back to their original types, which can lead to runtime errors if done incorrectly.

**Code:**

ArrayList nonGenericList = new ArrayList();

nonGenericList.Add("umer");

nonGenericList.Add(25);

nonGenericList.Add(true);

**Generic Collections:**

Generic collections, introduced in, are type-safe collections because they allow you to specify the type of elements they can contain at compile time.It use in data structures.

* **Type Safety:** With generic collections, the type of data being stored is known at compile time. This prevents type-related errors during runtime.
* **Performance:** Generic collections avoid the overhead of boxing and unboxing (the process of converting value types to reference types and vice versa), resulting in better performance.
* **Code Reusability:** Generics promote the creation of reusable classes, methods, and algorithms because they can work with multiple data types.
* **Compile-Time Checking:** Any mismatch in data types is caught by the compiler, ensuring code correctness.

**Code:**

List<string> names = new List<string>();

names.Add("umer");

names.Add("yasir");

names.Add("ali");

**Ans no 2:**

**Dynamic Form Entries:**

For Form Design entries based on user needs, where the number and type of entries are not known in advance, a List<T> is suitable.

Definition:  
A List<T> is a dynamic collection in C# that can grow or shrink in size dynamically. It is suitable for scenarios where we need to store a variable number of elements of a specific type.

**Example:**

List<string> userEntries = new List<string>();

userEntries.Add("Name: umer");

userEntries.Add("Age: 22");

userEntries.Add("Email: bcsf20a012@pucit.edu");

// ... we can add more entries as needed

**Form with Fixed Entries:**

For forms where the number and types of entries are fixed and do not change, we can use a collection that matches the fixed structure of our form. we might use **array**s or a collection of custom objects.

Here's why it's suitable:

* **Predictability:** When form entries are fixed, using arrays or a predefined collection structure provides clarity and predictability.
* **Type Safety:** we know exactly what types of data to expect, which leads to type safety in our code.
* **Validation:** Fixed form structures make it easier to define validation rules and constraints.