1. Describe the problem generics address.

Generics address the problem of type safety and code reusability. They allow you to define classes, methods, and data structures with a placeholder for the data type, which can be specified later when the class or method is instantiated or called. This reduces code duplication and eliminates the need for type casting, thereby preventing runtime errors and improving performance.

1. How would you create a list of strings, using the generic List class?

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

1. How many generic type parameters does the Dictionary class have?

two generic type parameters: one for the key and one for the value.

1. True/False. When a generic class has multiple type parameters, they must all match.

False. The type parameters in a generic class do not need to match; they can be different types.

1. What method is used to add items to a List object?

The ‘Add’ method is used to add items to a List object.

1. Name two methods that cause items to be removed from a List.

numbers.Remove(1);

numbers.RemoveAt(0);

1. How do you indicate that a class has a generic type parameter?

public class Class**<T>**

1. True/False. Generic classes can only have one generic type parameter.

False. Generic classes can have multiple generic type parameters.

1. True/False. Generic type constraints limit what can be used for the generic type.

True. Generic type constraints limit the types that can be used for the generic type, providing more control over the operations that can be performed on the type.

1. True/False. Constraints let you use the methods of the thing you are constraining to.

True. Constraints allow you to use the methods and properties of the type you are constraining to, ensuring that the generic type parameter meets certain requirements.

Generics codes:

1. Stack class:

using System;

using System.Collections.Generic;

namespace CustomStack

{

public class MyStack<T>

{

private List<T> \_items = new List<T>();

// Method to get the count of items in the stack

public int Count()

{

return \_items.Count;

}

// Method to pop an item from the stack

public T Pop()

{

if (\_items.Count == 0)

throw new InvalidOperationException("The stack is empty.");

T item = \_items[\_items.Count - 1];

\_items.RemoveAt(\_items.Count - 1);

return item;

}

// Method to push an item onto the stack

public void Push(T item)

{

\_items.Add(item);

}

}

1. Mylist

using System;

using System.Collections.Generic;

namespace CustomList

{

public class MyList<T>

{

private List<T> \_items = new List<T>();

// Method to add an element to the list

public void Add(T element)

{

\_items.Add(element);

}

// Method to remove an element at a specific index

public T Remove(int index)

{

if (index < 0 || index >= \_items.Count)

throw new ArgumentOutOfRangeException(nameof(index), "Index is out of range.");

T item = \_items[index];

\_items.RemoveAt(index);

return item;

}

// Method to check if the list contains a specific element

public bool Contains(T element)

{

return \_items.Contains(element);

}

// Method to clear the list

public void Clear()

{

\_items.Clear();

}

// Method to insert an element at a specific index

public void InsertAt(T element, int index)

{

if (index < 0 || index > \_items.Count)

throw new ArgumentOutOfRangeException(nameof(index), "Index is out of range.");

\_items.Insert(index, element);

}

// Method to delete an element at a specific index

public void DeleteAt(int index)

{

if (index < 0 || index >= \_items.Count)

throw new ArgumentOutOfRangeException(nameof(index), "Index is out of range.");

\_items.RemoveAt(index);

}

// Method to find an element at a specific index

public T Find(int index)

{

if (index < 0 || index >= \_items.Count)

throw new ArgumentOutOfRangeException(nameof(index), "Index is out of range.");

return \_items[index];

}

}

1. GenericRepository<T>

public interface IEntity

{

int Id { get; set; }

}

public interface IRepository<T> where T : IEntity

{

void Add(T item);

void Remove(T item);

void Save();

IEnumerable<T> GetAll();

T GetById(int id);

}

using System;

using System.Collections.Generic;

using System.Linq;

public class GenericRepository<T> : IRepository<T> where T : class, IEntity

{

private List<T> \_entities = new List<T>();

public void Add(T item)

{

if (item == null)

throw new ArgumentNullException(nameof(item));

\_entities.Add(item);

}

public void Remove(T item)

{

if (item == null)

throw new ArgumentNullException(nameof(item));

\_entities.Remove(item);

}

public void Save()

{

// In a database scenario, this might save changes to the database context.

// For in-memory, it might just be a no-op.

Console.WriteLine("Changes saved.");

}

public IEnumerable<T> GetAll()

{

return \_entities;

}

public T GetById(int id)

{

return \_entities.FirstOrDefault(e => e.Id == id);

}

}