GENERICS IN C#

Arctech Info Private Limited

BOXING & UNBOXING REVISITED

Boxing

- **`System.Object`** is the base class for all the classes in . Net Framework.
- Boxing - the process of converting a value type to object.
- When the CLR boxes a value type,
  - It wraps the value type into a new instance of **`System.Object`**
  - Stores the object on the heap

```
int i = 123;
// The following line boxes i.
object o = i;
```

UnBoxing

- Unboxing extracts the value type from the object.
- Boxing is implicit; unboxing is explicit.
- Boxing & UnBoxing are slow

```
object  o = 123;
i = (int)o;   // unboxing
```

C# VERSION 1.0

- Before C# 2.0, you could not have a standard algorithm to work with various types.
- The most common algorithm in programming are Collections like Stack, List, etc.

If you wanted to maintain two lists you have to create your classes yourself as below

```
public class StudentList
{
    // Write all the logic of StudentList
}

public class TeacherList
{
    // Write all the logic of StudentList

}
```

At that time .NET Team provided collection classes which stored objects, as below
Note: below is slow as it involves Boxing/Unboxing

```
ArrayList students = new ArrayList();
students.Add(100);

// But since the Add method accepts objects
// The below works which is a huge design issue
// So developer had to be careful not to do this
students.Add("Wow!!");
```

GENERICS IN C# 2.0

- Generics introduces the concept of type parameters to .NET
- By using a generic type parameter, you can create a single class which other classes can use without boxing/unboxing operations
- Since generics are resolved at compile time, it is very fast.
- .NET Team introduced new collections like `List<T>`, `Dictionary<TKey, TValue>`, etc.

```
var ages = new List<int>();
ages.Add(65);

// Compile Error
// cannot convert from 'string' to 'int'
ages.Add("Wow!!");

var teachers = new Dictionary<string, Teacher>();
teachers.Add("A101",
    new Teacher("Raman Gujral", 50000));
teachers.Add("A102",
    new Teacher("Madhumita Sharma", 75000));
```

ADVANTAGE OF GENERICS

- Allows you to write code which are type-safe, i.e., a `List<string>` is guaranteed to be a list of strings.
- Are you trying to put an int into that list of strings?
  - With `List<string>` you get an immediate compile time error
  - With `ArrayList` you get a delayed and less obvious runtime error.
- Faster as it
  - avoids boxing/unboxing in case of value types or
  - casting from objects in case of reference types.
- Allows you to write code which is applicable to many types with the same underlying behavior
- E.g., a `Dictionary<string, int>` uses the same underlying code as a `Dictionary<DateTime, double>`

MORE FEATURES OF GENERICS

- Generic classes and methods combine reusability, type safety, and efficiency in a way that their non-generic counterparts cannot.
- Generics are most frequently used with collections and the methods that operate on them.
- The **`System.Collections.Generic`** namespace contains several generic-based collection classes.
- The non-generic collections, such as **`ArrayList`** are not recommended and are maintained for compatibility purposes.

YOUR OWN GENERIC METHOD

Before Generics

```
void SwapNumber(ref int leftSide, ref int rightSide)
{
    int temp;
    temp = leftSide;
    leftSide = rightSide;
    rightSide = temp;
}

void SwapString(ref string leftSide, ref string rightSide)
{
    string temp;
    temp = leftSide;
    leftSide = rightSide;
    rightSide = temp;
}
```

A generic method is a method that is declared with type parameters, as follows:

```
void Swap<T>(ref T leftSide, ref T rightSide)
{
    T temp;
    temp = leftSide;
    leftSide = rightSide;
    rightSide = temp;
}
void Test()
{
    int a = 10, b = 20;
    Swap<int>(ref a, ref b);
    Console.WriteLine($"a={a} | b={b}");
}
```

YOU OWN GENERIC CLASS

```
public class DatabaseTable<T>
{
    public void Insert(T obj)
    {
    }

    public void Update(T obj)
    {
    }

    public void Delete(T obj)
    {
    }
}
```

```
void TestDatabaseTable()
{
    var obj1 = new DatabaseTable<Employee>();
    obj1.Insert(
        new Employee(101, "Raman Gujral", 25000));

    var obj2 = new DatabaseTable<Department>();
    obj2.Insert(
        new Department(10, "Marketing"));
    obj2.Delete(
        new Department(20, "Sales"));
}
```