Generics

28 January 2022 09:28

CLR Generics: It is syntactical support offered by the intermediate language for implementing *type-safe* code patterns which can be reused with different *data-types*. It enables a high-level language (such as C#) compiler to identify matching data-types in a declaration and to eliminate any unnecessary type conversion (such as casting, boxing and unboxing).
A generic declaration contains at least one *type argument* which is open to substitution with a known type and it is replaced by this type at runtime (reification). A type argument T is treated as System.Object type at compile time and as such it can be substituted by any type unless it appears in the declaration with following constraints.

1. **T: struct** - T can only be substituted by a value type
   T: class - T can only be substituted by a reference type
2. **T: R** - T can only be substituted by a type which supports implicit conversion to (inherits from) reference type R and as such members of R can be applied to T.
3. **T: new()** - T can only be substituted by a type which supports a parameter-less constructor and as such new operator can be applied to T with zero arguments.

Variance in Generics: By default a generic type G *is invariant* over its type argument T meaning G<V> cannot be converted to G<U> irrespective of any relationship between types U and V. A generic interface I with type argument T can be defined in

1. **Covariant form as I<out T>** to indicate that T does not appear as a parameter type in members of I and as such I<V> can be converted to I<U> if U and V are reference types such that V supports implicit conversion to (inherits from) U.
2. **Contravariant form as I<in T>** to indicate that T does not appear as a return type, ref or out parameter type in members of I and as such I<V> can be converted to I<U> if U and V are reference types such that U supports implicit conversion to (inherits from) V.

Generic Collections: The BCL includes System.Collections.Generic namespace which provides support for generic collections. It defines **ICollection**<V> interface which extends **IEnumerable**<V> interface and is extended by

1. **IList**<V> - it specifies support for collecting indexed values and for retrieving them is a sequential manner. It is implemented by List<V>
2. **ISet**<K> - it specifies support for collecting unique keys and for retrieving them in order of their values. It is implemented by HashSet<K> and SortedSet<K>
3. **IDictionary**<K, V> - it specifies support for collecting pairs each containing a unique key and a value mapped to that key. It is implemented by Dictionary<K, V>, SortedList<K, V> and SortedDictionary<K, V>