Generics

What are generics?

- Allow creation of type-safe data structure.
- Reuse data processing algorithms without duplicating type-specific code
- They are not only constructs of C# but are the constructs of CLR.
- Can be defined with class and struct

HCL

Creating generic class

```
using System;
class Stack<T>{
    T[]  items;
    int top=-1;
 public Stack()    {  items = new
  T[10];    }

    public void Push(T item)    {
  if(top<9)
  items[++top]=item;
  else
  System.Console.WriteLine("full");
}
```

Convention is Generic type name should be prefixed with a letter T.

HCL

```
public T Pop(){
        if(top>0)     return items[top--];
        else{
        System.Console.WriteLine("empty");
        return null;
        }
}
}
```

Test.cs(17,9): error CS0403: Cannot convert null to type parameter 'T' because it could be a value type. Consider using 'default(T)' instead.

```
return default(T);
```

`default()` operator returns the default value of a type.

HCL

OR throw an exception

```
public T Pop(){
     if(top>0)     return items[top--];
     else{
     System.Console.WriteLine("empty");
     throw new
InvalidOperationException("Cannot pop
an empty stack");
     }
}
}
```

HCL

Using generic class

```
public class Test{

public static void Main(){

Stack<int> stack = new Stack<int>();

stack.Push(1);

stack.Push(2);

int number = stack.Pop();

System.Console.WriteLine(number);

}}
```

HCL

Advantages of generics

- Type Safety:

Collection class we will cover ahead

```
ArrayList list =new ArrayList();

list.Add(4);

list.Add("str");


ArrayList<int> list =new ArrayList();

list.Add(4);

list.Add("str");// compile time error
```

HCL

- Performance:

```
ArrayList list =new ArrayList();

list.Add(4);// boxing

…

int i=(int) list[0]; // unboxing


List<int> list =new List<int>;

list.Add(4);// no boxing

…

int i=list[0]; // no unboxing
```

HCL

- Binary Code reuse
  - Generic types can be defined in one language and can be used from any other .NET language.

Disadvantage

- Code Bloat:
  - When generic classes are compiled by the JIT to native code, an new class for every specific value type is created.

HCL

Multiple Generic Types

- Defining more than one generic type
  - ```
    class Map<K,T>{

    …

    T getObject(K key){    …    }

    void add(K key, T value){…}

    }
    ```
- Instantiating
  - ```
    Map<int,string> m= new

    Map<int,string>();
    ```

HCL

Generic type aliasing

- Used to alias a particular combination of specific types
-
```
using MyMap = Map<int,string>; class
Client {
  static void Main(string[] args) {
    MyMap m = new MyMap();
    …
}
```

Comparison of generics

- Lets add a method to find if an element exist in the stack.

```
class Stack{
…
bool find(T t){
for(int i=0;i<=top;i++)
if(t==items[i]) return true;
return false;
}
```

Stack1.cs(23,4): error CS0019: Operator '==' cannot be applied to operands of type 'T' and 'T'

```
if(t.Equals(items[i]))
```

Ok but how to make sure if equals has been implemented properly.

?

HCL

Generic Constraints

- This happened because the compiler compiles the generic code in an IL independent way irrespective of argument type the client passes.
- To instruct the compiler to use a particular type for generics which the client is expected to use, generic constraint can be specified.

HCL

Types of Generic Constraints

- Derivation constraint:
  - tells the compiler that the generic type parameter derives from a base type such an interface or a particular base class.
- Default constructor constraint:
  - Tells the compiler that the generic type parameter exposes a default public constructor (a public constructor with no parameters).
- Reference/value type constraint:
  - constrains the generic type parameter to be a reference or a value type.

HCL

Derivation constraint

- To solve the comparison problem using **IComparable**
  - Use derivation constraint to make sure that template type is **IComparable**.
  - Use **CompareTo**() to compare

```
class Stack2<T> where T :
  IComparable{
…
public bool find(T t){
for(int i=0;i<=top;i++){
if(t.CompareTo(items[i])==0) return
  true;}
return false;
}}}
```

after the actual derivation list of the generic class

HCL

Multiple Derivation constraint

- multiple interfaces on the same generic type parameter
  - **`public class Map<K,T> where K : IComparable, ICloneable`**
- constraints for every generic type parameter your class uses
  - **`public class LinkedList<K,T> where K : IComparable<K> where T : ICloneable`**
- constrain both a base class and one or more interfaces
  - **`public class LinkedList<K,T> where K : MyClass,ICloneable`**
- **`public class MyClass<T,U> where T : U`** →wrong

HCL

Constructor constraint

- Suppose there is a need to create an entry into stack in the Stack constructor.

```
public Stack2()    {
 items = new T[10];
 items[0] =new T();
}
```

error CS0304: Cannot create an instance of the variable type 'T' because it does not have the new() constraint

- Specify new constraint
  - `class Stack2<T>  where  T :  new(){`

```
public Stack2()    {
  items = new T[10];
  items[0] =new T();
}
```

If there are other constraints **new** must be the last.

```
class Stack2<T>  where
T : IComparable, new(){
```

Reference/Value Type Constraint

- constrain a generic type parameter to be a value type (such as an int, a bool, and enum) or any custom structure using the **struct** constraint
  - `public class MyClass<T> where T : struct {...}`
- you can constrain a generic type parameter to be a reference type (a class) using the **class** constraint
  - `public class MyClass<T> where T : class {...}`

HCL

Generics and Casting

- Implicit casting is possible only from generic type parameters to Object, or to constraint-specified types
  - **`class MyClass<T> where T : YourClass, ICloneable {`**
    `f(T t){`
    `YourClass y=t;`
    `ICloneable i=t;} …}`
- To force a cast from a generic type parameter to any other type using a temporary Object variable
  - **`void f(T t){ object temp = t;`**
    `SomeClass obj = (SomeClass)temp; }`
- Usage of **`is`** and **`as`**:

`if(t is int), if(t is Stack<int>)`

`string str = t as string;`

HCL

Inheritance in generics

- `public class BaseClass<T> {...} public class SubClass : BaseClass<int> {...}`
- `public class SubClass<T> : BaseClass<T> {...}`
- `public class BaseClass<T> where T : ISomeInterface {...} public class SubClass<T> : BaseClass<T> where T : ISomeInterface {...}`

HCL

Subclass methods

- ```
  public class BaseClass<T> {
  public virtual T SomeMethod() {...} }
  public class SubClass: BaseClass<int> {
  public override int SomeMethod() {...} }
  ```
- ```
  public class SubClass<T>: BaseClass<T> {
  public override T SomeMethod() {...} }
  ```

HCL

Generic methods

- Defining method specific generic types

```
using System;
class Display{
public static void display<T>(T[] t){
  for(int i=0;i<t.Length;i++)
  System.Console.WriteLine(t[i]);
}
public static void Main(){
string [] s= {"April", "May", "June",
  "July"};
Display.display<string>(s);
}}
```

Omitting the type-spec is also OK

HCL

Generic delegates

- A delegate defined in a class can take advantage of the generic type parameter of that class.
- Otherwise, like classes, structs, and methods, delegates can define their own generic type parameters .

HCL

Generic delegates inside a class

- ```
  public class MyClass<T> {
  public delegate void GDelegate(T t);
  public void f(T t) {...}
  }
  ```
- Instantiating

```
MyClass<int> obj = new
MyClass<int>();
MyClass<int>.GDelegate gd;
gd = new
MyClass<int>.GDelegate(obj.f);
gd(12);
```

Or simply `gd=obj.f;`

delegate inference

HCL
```

Generic delegates outside a class

- ```
  public delegate void GDel<T>(T t);

   public class MyClass {

  public void f(string s) {...}

  }
  ```
- ```
  MyClass o = new MyClass();

  GDel<string> s;

  s = new GDel<string>(o.f);

  s("hello");
  ```

HCL

System.Array

- The **System.Array** type is extended with many generic static methods.
- The static generic methods all work with the following four generic delegates defined in the **System** namespace
- **public delegate void Action<T>(T t);**
- **public delegate int Comparison<T>(T x, T y);**
- **public delegate U Converter<T, U>(T from);**
- **public delegate bool Predicate<T>(T t);**

HCL

Comparison in sort() method

- `public static void Sort<T>(T[] array,Comparison<T> comparison)`
- This method works with the delegate

```
public delegate int Comparison<T>(Tx, Ty);
```

HCL

Example using sort()

```
using System;
public class Employee{
private int empID;
private string empName;
public Employee(string nm, int id){
this.empID=id;
this.empName=nm;
}
public string Name{
  get{return empName;}
  set{
  System.Console.WriteLine("SET");
  if(value!=null)
  empName=value;     }}
public int ID{
  get{return empID;}
  set{if(value!=0)
  empID=value; }}
```

HCL

```
static void Main(){

Employee[] e= {new Employee("Neeta" ,12), new
Employee("Surya",11),new Employee("Smita" ,3) };

Array.Sort(e, delegate(Employee e1, Employee e2){

      return e1.Name.CompareTo( e2.Name);
 });

for(int i=0;i<e.Length;i++)
System.Console.WriteLine("Name is {0},
            ID is {1}" ,e[i].Name,e[i].ID);
}
}
```

or

```
return e1.ID - e2.ID;
```

HCL

Action in ForEach() method

- `public static void ForEach<T>(T[] array, Action<T> action);`

- Uses delegate

  - `public delegate void Action<T>(T t);`

HCL

```
public class Employee{
…
override public String ToString(){
return empName+" ( "+ empID +" ) ";
}
static void Main(){
Employee[] e= {…};
Array.Sort(e, delegate(Employee e1,
Employee e2){
return e1.Name.CompareTo( e2.Name);
 });

Array.ForEach(e, Console.WriteLine);}}
```

HCL

Converter in ConvertAll()

- `public static U[] ConvertAll<T,U>(T[] array, Converter<T,U> converter);`
- Uses the following delegate
  - `public delegate U Converter<T, U>(T from);`
- This method is used if an array of one type should be converted to an array of another type.

HCL

- Suppose we have a Trainee class which also has Name and ID as attributes apart from other specific methods of its own. The following example demonstrates the conversion of Employee array into Trainee array:

```
class Test{
static void Main(){
Employee[] e= {new Employee("Neeta"
  ,12), new Employee("Surya",11),new
  Employee("Smita" ,3) };
Trainee[] t=
  Array.ConvertAll<Employee,Trainee>(e,de
  legate(Employee e1){
return new Trainee(e1.Name, e1.ID);
});
Array.ForEach(t, Console.WriteLine);}}
```

Predicate with find

- `public static T Find<T>(T[] array,Predicate<T> match);`
- `public static T[] FindAll<T>(T[] array, Predicate<T> match);`
- `public static int FindIndex<T>(T[] array, Predicate<T> match);`
- All the find methods use the delegate
  - `public delegate bool Predicate<T>(T t);`

HCL

```
static void Main(){
Employee[] e= {new Employee("Neeta"
,12), new Employee("Surya",11),new
Employee("Smita" ,3) };
Employee[] s=Array.FindAll(e,
delegate(Employee e1){
     return e1.Name.StartsWith("S");
});
Array.ForEach(s, Console.WriteLine);
}
```

HCL