```
public interface I<T>
{
    T Value { get; }
}

public static class IndexingClass
{
    public static Dictionary<T,IEnumerable<I<T>?>> Indexing<T>(
         this IEnumerable<I<T>?> s) where T : Enum
    {
        Dictionary<T, IEnumerable<I<T>?>> Indexing_Aux()
        {
            using (var it = s.GetEnumerator())
            {
                //Inizializzazione
                Dictionary<T, IEnumerable<I<T>?>> dict = new();
                var types = Enum.GetValues(typeof(T));
                foreach (var type in types)
                {
                    dict.Add((T)type, new List<I<T>>());
                }
                while (it.MoveNext())
                {
                    if (null == it.Current)
                     throw new ArgumentNullException(nameof(it.Current));
                    var list = dict[it.Current.Value];
                    dict[it.Current.Value] = list.Append(it.Current);

                }
                return dict;
            }
        }

        if(null == s) throw new ArgumentNullException(nameof(s));
        return Indexing_Aux();
    }
}

public class IndexingTest
{
    public enum Day{Mo,Tu,We,Th,Fr  };

    public enum Color{White,Grey,Black}

    public class C<T> : I<T>
    {
        public T Value { get; set; }

        public C(T value)
        {
            Value = value;
        }

        public override bool Equals(object? obj)
        {
            return obj is C<T> elem &&
                   Value.Equals(elem.Value);
        }

        public override int GetHashCode()
        {
            return Value.GetHashCode();
        }
    }
```

```
        [Test]
        public void Test1()
        {
            Assert.That(() => new[]
                {
                    new C<Day>(Day.Mo),
                    new C<Day>(Day.Fr),
                    null,
                    new C<Day>(Day.Fr),
                    new C<Day>(Day.Fr)
                }.Indexing().ToArray(),
                Throws.InstanceOf<ArgumentNullException>());
        }

        [Test]
        public void Test2()
        {
            var ris = new[]
            {
                new C<Day>(Day.Mo),
                new C<Day>(Day.Mo),
                new C<Day>(Day.We),
                new C<Day>(Day.Mo),
                new C<Day>(Day.Fr),
                new C<Day>(Day.We),
            }.Indexing();

            Assert.Multiple(() =>
                {
                    Assert.That(ris[Day.Mo], Is.EqualTo(new[] { new C<Day>(Day.Mo),
                   new C<Day>(Day.Mo), new C<Day>(Day.Mo) }.ToList()));
                    Assert.That(ris[Day.Tu], Is.EqualTo(Array.Empty<C<Day>>()));
                    Assert.That(ris[Day.We], Is.EqualTo(new[] { new C<Day>(Day.We),
                   new C<Day>(Day.We) }.ToList()));
                    Assert.That(ris[Day.Th], Is.EqualTo(Array.Empty<C<Day>>()));
                    Assert.That(ris[Day.Fr], Is.EqualTo(new[] { new C<Day>(Day.Fr)
}.ToList()));
                }

            );
        }

        [TestCase(5)]
        public void colors(int howMany)
        {
            IEnumerable<C<Color>> GenSeq()
            {
                for (int i = 0; i < howMany; i++)
                {
                    yield return new C<Color>(Color.White);
                    yield return new C<Color>(Color.Grey);
                    yield return new C<Color>(Color.Black);
                }
            }

            var ris = GenSeq().Indexing();
            Assert.Multiple(() =>
            {
                Assert.That(ris[Color.White].Count(),Is.EqualTo(howMany));
                Assert.That(ris[Color.Grey].Count(), Is.EqualTo(howMany));
                Assert.That(ris[Color.Black].Count(), Is.EqualTo(howMany));
            });
        }
```