1.1 ./csharp/Platform.Interfaces/ICounter[TResult, TArgument].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a counter that requires an argument to perform a count.</para>
    /// <para>Определяет счётчик, которому требуется аргумент для выполнения подсчёта.</para>
    /// </summary>
    /// <typeparam name="TArgument">
    /// <para>The argument type.</para>
    /// <para>Тип аргумента.</para>
    /// </typeparam>
    /// <typeparam name="TResult">
    /// <para>The count result type.</para>
    /// <para>Тип результата подсчёта.</para>
    /// </typeparam>
    public interface ICounter<out TResult, in TArgument>
    {
        /// <summary>
        /// <para>Performs a count.</para>
        /// <para>Выполняет подсчёт.</para>
        /// </summary>
        /// <param name="argument">
        /// <para>The argument.</para>
        /// <para>Аргумент.</para>
        /// </param>
        /// <returns>
        /// <para>The count result.</para>
        /// <para>Результат подсчёта.</para>
        /// </returns>
        TResult Count(TArgument argument);
    }
}
```

1.2 ./csharp/Platform.Interfaces/ICounter[TResult].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a counter.</para>
    /// <para>Определяет счётчик.</para>
    /// </summary>
    /// <typeparam name="TResult">
    /// <para>The count result type.</para>
    /// <para>Тип результата подсчёта.</para>
    /// </typeparam>
    public interface ICounter<out TResult>
    {
        /// <summary>
        /// <para>Performs a count.</para>
        /// <para>Выполняет подсчёт.</para>
        /// </summary>
        /// <returns>
        /// <para>The count result.</para>
        /// <para>Результат подсчёта.</para>
        /// </returns>
        TResult Count();
    }
}
```

1.3 ./csharp/Platform.Interfaces/ICriterionMatcher.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a criterion matcher, that contains a specific method for determining
    ↪  whether the argument matches the criterion or not.</para>
    /// <para>Определяет объект который проверяет соответствие критерию и содержит конкретный
    ↪  метод для определения, соответствует ли аргумент критерию или нет.</para>
    /// </summary>
    /// <typeparam name="TArgument">
    /// <para>Argument type.</para>
    /// <para>Тип аргумента.</para>
    /// </typeparam>
    public interface ICriterionMatcher<in TArgument>
    {
        /// <summary>
        /// <para>Determines whether the argument matches the criterion.</para>
        /// <para>Определяет, соответствует ли аргумент критерию.</para>
```

```
        /// </summary>
        /// <param name="argument">
        /// <para>The argument.</para>
        /// <para>Аргумент.</para>
        /// </param>
        /// <returns>
        /// <para>A value that determines whether the argument matches the criterion.</para>
        /// <para>Значение, определяющие соответствует ли аргумент критерию.</para>
        /// </returns>
        bool IsMatched(TArgument argument);
    }
}
```

1.4 ./csharp/Platform.Interfaces/IFactory.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a factory that produces instances of a specific type.</para>
    /// <para>Определяет фабрику, которая производит экземпляры определенного типа.</para>
    /// </summary>
    /// <typeparam name="TProduct">
    /// <para>Type of produced instances.</para>
    /// <para>Тип производимых экземпляров.</para>
    /// </typeparam>
    public interface IFactory<out TProduct>
    {
        /// <summary>
        /// <para>Creates an instance of <typeparamref name="TProduct"/> type.</para>
        /// <para>Создает экземпляр типа <typeparamref name="TProduct"/>.</para>
        /// </summary>
        /// <returns>
        /// <para>The instance of <typeparamref name="TProduct"/> type.</para>
        /// <para>Экземпляр типа <typeparamref name="TProduct"/>.</para>
        /// </returns>
        TProduct Create();
    }
}
```

1.5 ./csharp/Platform.Interfaces/IProperties.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a properties operator that is able to get or set values of properties of a
    ↪  object of a specific type.</para>
    /// <para>Определяет оператор свойств, который может получать или устанавливать значения
    ↪  свойств объекта определенного типа.</para>
    /// </summary>
    /// <typeparam name="TObject">
    /// <para>Object type.</para>
    /// <para>Тип объекта.</para>
    /// </typeparam>
    /// <typeparam name="TProperty">
    /// <para>Property reference type.</para>
    /// <para>Тип ссылки на свойство.</para>
    /// </typeparam>
    /// <typeparam name="TValue">
    /// <para>Property value type.</para>
    /// <para>Тип значения свойства.</para>
    /// </typeparam>
    public interface IProperties<in TObject, in TProperty, TValue>
    {
        /// <summary>
        /// <para>Gets the value of the property in the specified object.</para>
        /// <para>Получает значение свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="object">
        /// <para>The object reference.</para>
        /// <para>Ссылка на объект.</para>
        /// </param>
        /// <param name="property">
        /// <para>The property reference.</para>
        /// <para>Ссылка на свойство.</para>
        /// </param>
        /// <returns>
        /// <para>The value of the property.</para>
        /// <para>Значение свойства.</para>
        /// </returns>
```

```
        TValue GetValue(TObject @object, TProperty property);

        /// <summary>
        /// <para>Sets the value of a property in the specified object.</para>
        /// <para>Устанавливает значение свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="object">
        /// <para>The object reference.</para>
        /// <para>Ссылка на объект.</para>
        /// </param>
        /// <param name="property">
        /// <para>The property reference.</para>
        /// <para>Ссылка на свойство.</para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para>Значение.</para>
        /// </param>
        void SetValue(TObject @object, TProperty property, TValue value);
    }
}
```

1.6 ./csharp/Platform.Interfaces/IProperty.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a specific property operator that is able to get or set values of that property.</para>
    /// <para>Определяет оператор определённого свойства, который может получать или устанавливать его значения.</para>
    /// </summary>
    /// <typeparam name="TObject">
    /// <para>Object type.</para>
    /// <para>Тип объекта.</para>
    /// </typeparam>
    /// <typeparam name="TValue">
    /// <para>Property value type.</para>
    /// <para>Тип значения свойства.</para>
    /// </typeparam>
    public interface IProperty<in TObject, TValue> : ISetter<TValue, TObject>, IProvider<TValue, TObject>
    {
    }
}
```

1.7 ./csharp/Platform.Interfaces/IProvider[TProvided, TArgument].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines the provider of objects/values for which an argument must be specified.</para>
    /// <para>Определяет поставщика объектов/значений, для получения которых необходимо указать аргумент.</para>
    /// </summary>
    /// <typeparam name="TProvided">
    /// <para>Type of provided objects/values.</para>
    /// <para>Тип предоставляемых объектов/значений.</para>
    /// </typeparam>
    /// <typeparam name="TArgument">
    /// <para>Argument type.</para>
    /// <para>Тип аргумента.</para>
    /// </typeparam>
    public interface IProvider<out TProvided, in TArgument>
    {
        /// <summary>
        /// <para>Provides an object(s)/value(s).</para>
        /// <para>Предоставляет объект(ы)/значение(я).</para>
        /// </summary>
        /// <param name="argument">
        /// <para>The argument required to acquire the object(s)/value(s).</para>
        /// <para>Аргумент, необходимый для получения объекта(ов)/значения(ий).</para>
        /// </param>
        /// <returns>
        /// <para>The object(s)/value(s).</para>
        /// <para>Объект(ы)/значение(я).</para>
        /// </returns>
        TProvided Get(TArgument argument);
```

```
    }
}
```

1.8 ./csharp/Platform.Interfaces/IProvider[TProvided].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines the provider of objects/values.</para>
    /// <para>Определяет поставщика объектов/значений.</para>
    /// </summary>
    /// <typeparam name="TProvided">
    /// <para>Type of provided object/value.</para>
    /// <para>Тип предоставляемого объекта/значения.</para>
    /// </typeparam>
    public interface IProvider<out TProvided>
    {
        /// <summary>
        /// <para>Provides an object(s)/value(s).</para>
        /// <para>Предоставляет объект(ы)/значение(я).</para>
        /// </summary>
        /// <returns>
        /// <para>The object(s)/value(s).</para>
        /// <para>Объект(ы)/значение(я).</para>
        /// </returns>
        TProvided Get();
    }
}
```

1.9 ./csharp/Platform.Interfaces/ISetter[TValue, TArgument].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines an setter that requires an argument to set the passed value as a new state.</para>
    /// <para>Определяет установщик, которому для установки переданного значения в качестве нового состояния требуется аргумент.</para>
    /// </summary>
    /// <typeparam name="TValue">
    /// <para>Type of set value.</para>
    /// <para>Тип устанавливаемого значения.</para>
    /// </typeparam>
    /// <typeparam name="TArgument">
    /// <para>The argument type.</para>
    /// <para>Тип аргумента.</para>
    /// </typeparam>
    public interface ISetter<in TValue, in TArgument>
    {
        /// <summary>
        /// <para>Sets the value of a specific property in the specified object.</para>
        /// <para>Устанавливает значение определённого свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="argument">
        /// <para>The argument.</para>
        /// <para>Аргумент.</para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para>Значение.</para>
        /// </param>
        void Set(TArgument argument, TValue value);
    }
}
```

1.10 ./csharp/Platform.Interfaces/ISetter[TValue].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines an setter that sets the passed value as a new state.</para>
    /// <para>Определяет установщик, который устанавливает переданное значение в качестве нового состояния.</para>
    /// </summary>
    /// <typeparam name="TValue">
    /// <para>Type of set value.</para>
    /// <para>Тип устанавливаемого значения.</para>
    /// </typeparam>
    public interface ISetter<in TValue>
    {
```

```
        /// <summary>
        /// <para>Sets the value of a specific property in the specified object.</para>
        /// <para>Устанавливает значение определённого свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para>Значение.</para>
        /// </param>
        void Set(TValue value);
    }
}
```

1.11 ./csharp/Platform.Interfaces.Tests/InterfacesTests.cs

```
using Xunit;

#pragma warning disable CS0168 // Variable is declared but never used
#pragma warning disable CS0219 // Variable is assigned but its value is never used

namespace Platform.Interfaces.Tests
{
    /// <summary>
    /// <para>
    /// Represents the interfaces tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class InterfacesTests
    {
        /// <summary>
        /// <para>
        /// Tests that build test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void BuildTest()
        {
            ICounter<int, int> c1 = null;
            ICounter<int> c2 = null;
            ICriterionMatcher<int> cm1 = null;
            IFactory<int> f1 = null;
            IProperties<int, int, int> p1 = null;
            IProperty<int, int> p2 = null;
            IProvider<int, int> p3 = null;
            IProvider<int> p4 = null;
            ISetter<int, int> s1 = null;
            ISetter<int> s2 = null;
        }
    }
}
```

Index