**Refactoring Documentation for Project RESTSharp**

**Presented by Team Halla**

**Code Refactoring:**

* + Placed curly braces around if-block, for- and foreach-loops when missing
  + Formatted the curly braces **{** and **}** according to the best practices for the

C# language

* Character casing: variables and fields made **camelCase**; types and methods

made **PascalCase**. All constants renamed using **PascalCase**.

* Placed empty line between methods, properties, classes etc. Following the

best practices of "High Quality Code"

* Removed all unneeded empty lines, e.g. in all classes in RestSharp project,

as well as RestSharp.Tests and RestSharp.IntegrationTests projects

* Class members ordered in correct order (fields, constructors, properties,

public static methods, public non-static methods, private static methods, private non-static methods)

* Always used built-in type aliases (string, int, long etc.) instead of String,

Int32, Int64, etc.

* All using directives placed inside the namespaces
* .this typed in front of all class members (fields, properties, methods)
* Placed access modifier on all classes, methods, etc.
* Renamed many variables following the best practices of HQC
* Files with more than one class parted in more files. Each file contains only

one class now

**Bug Fixing and Unit Tests:**

1. Fixed failing unit tests by uncommenting foreach-loop in

PopulateListFromElements method in file Deserializers/XmlDeserializer.cs

1. Fix remaining failing unit tests by applying a fix in property HasFiles in file Http.cs
2. Implemented all 11 TO-DO tests in JsonTests.cs
3. Implemented some more tests for Deserializers/XmlDeserializer class
4. Implemented tests for Extensions/StringExtensions class

**Code Documenting:**

* Documented methods, properties, etc of class HttpPostParameter
* Documented interface ISerializer
* Documented interface IAuthenticator
* Documented interface IHttpFactory

**SOLID Principles:**

1. Single Responsibility Principle:
   * **DotNetXmlDeserializer** class
   * **XmlDeserializer** class
2. Open / Closed Principle:
   * Lots of classes which implement interfaces (IDeserializer, IAuthenticator, ISerializer, etc.) are open to extensions
3. Liskov Substitution Principle
   * Class **WebParameter** inherits the class **WebPair**.

There is parameter **IEnumerable<WebPair> parameters** in constructor of class **WebParameterCollection**. It can be both **IEnumerable<WebPair>** and **IEnumerable<WebParameter>**

1. Interface Segregation Principle
   * **IDeserializer**, **IAuthenticator**, **IHttpFactory**, **ISerializer** are all small, offer only a few methods and properties and can be easily used when needed without leaving unimplemented methods
2. Dependency Inversion Principle
   * …

**Design Patterns:**

1. Creational Design Patterns
   * **Factory Method**

Generic class **SimpleFactory<T>** which implements **IHttpFactory**

1. Structural Design Patterns
   * **Façade**

public RestRequest()

{

this.Parameters = new List<Parameter>();

this.Files = new List<FileParameter>();

this.XmlSerializer = new XmlSerializer();

this.JsonSerializer = new JsonSerializer();

this.OnBeforeDeserialization = r => { };

}

1. Behavioral Design Patterns
   * ...