A SOLID design in CACHÉ Object

The Requirement….

Route	Method Type	Sample Request	Sample Response

http://dummy.restapiexample.com/api/v1/create	POST	{
		"name":"test",
		"salary":"123",
		"age":"23“
		}
			{
			"name":"test",
			"salary":"123",
			"age":"23",
			"id":"719“
			}

All we need to do is create a utility which will :

- Call the rest API to create record
- Display the results

I don’t want to see the Output, but This is dumbest code I ever seen!! Everything is written in a single Class, VERY RISKY!! Don’t you know about any Design Principles

Code is ready for your review and you can also check the output. The requirement is solved in a single class having 56 Lines!!

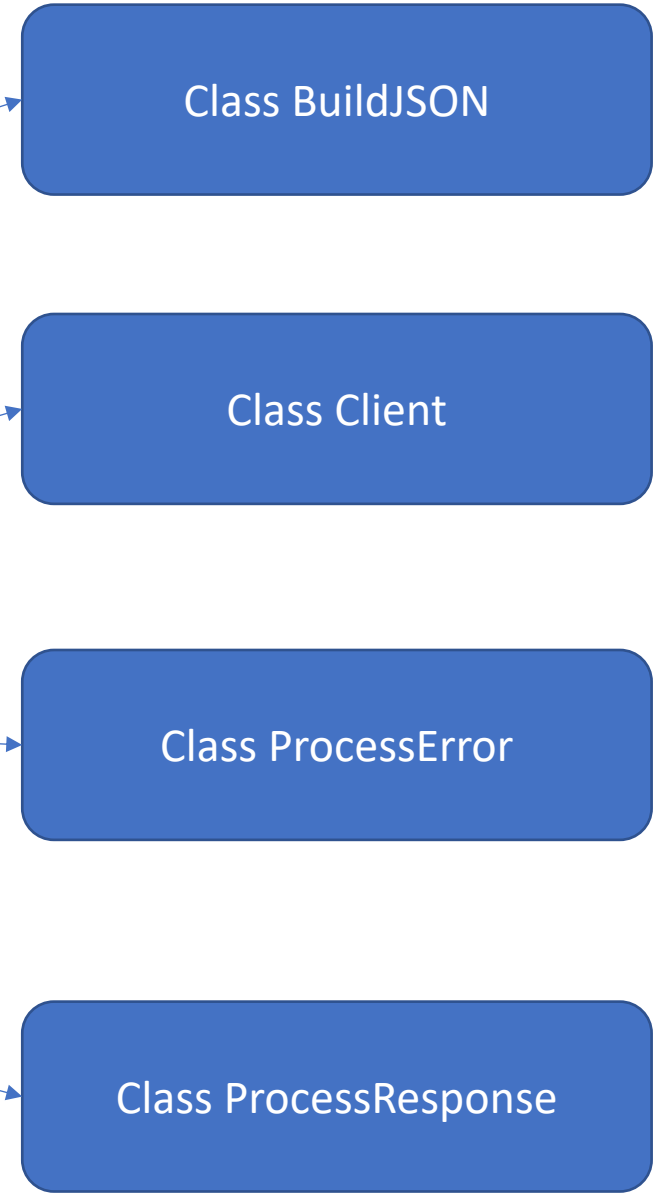
Haha, I am the greatest DEVELOPER!!

I heard about **Solid Principles**!! I need to revisit my code again with a different approach.

Single-responsibility Principle

A class should have one and only one reason to change, meaning that a class should have only one job.

```
Class CreateEmployee
{

        ClassMethod Execute(params) As %Status
        {

            //Request JSON

            //Rest Client, Build Request Body and POST

            //Handle Errors

            // write Response

        }
}
```



ADAPTIVE

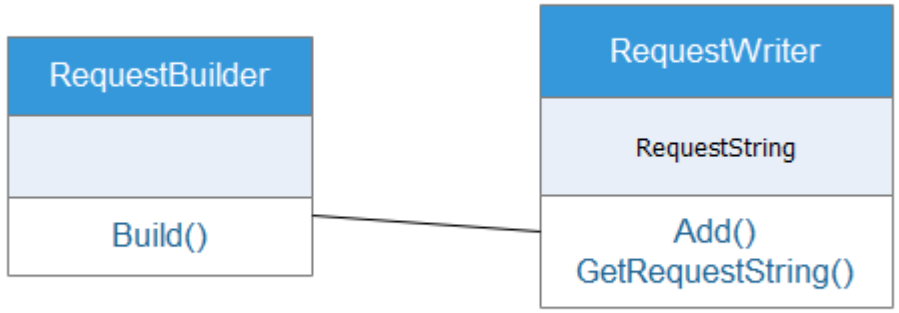
MAINTAINABLE

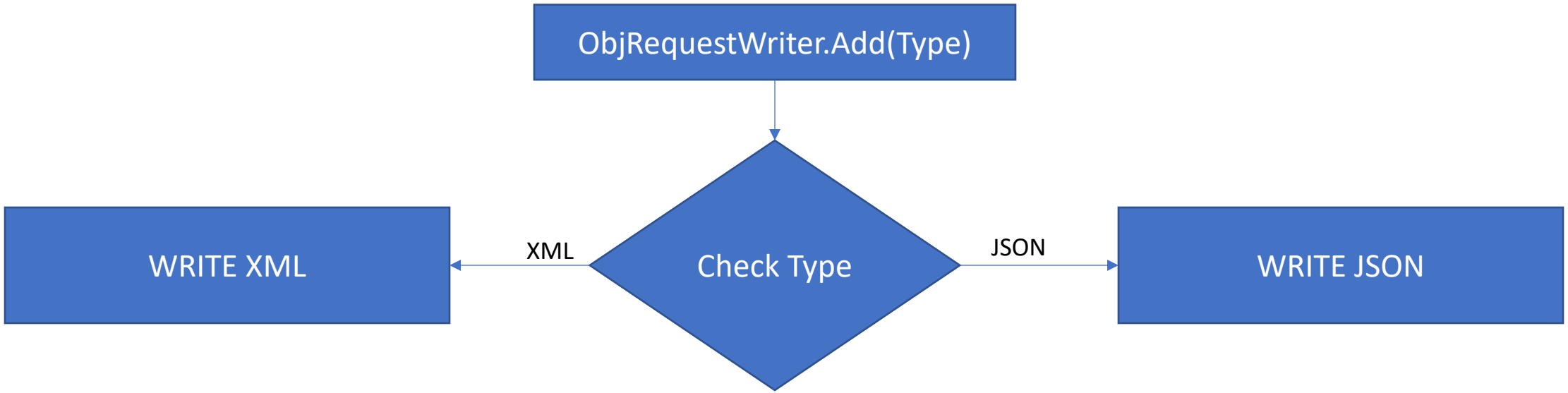
REQUIREMENT

V2 API

Accept XML
not JSON

Current Design

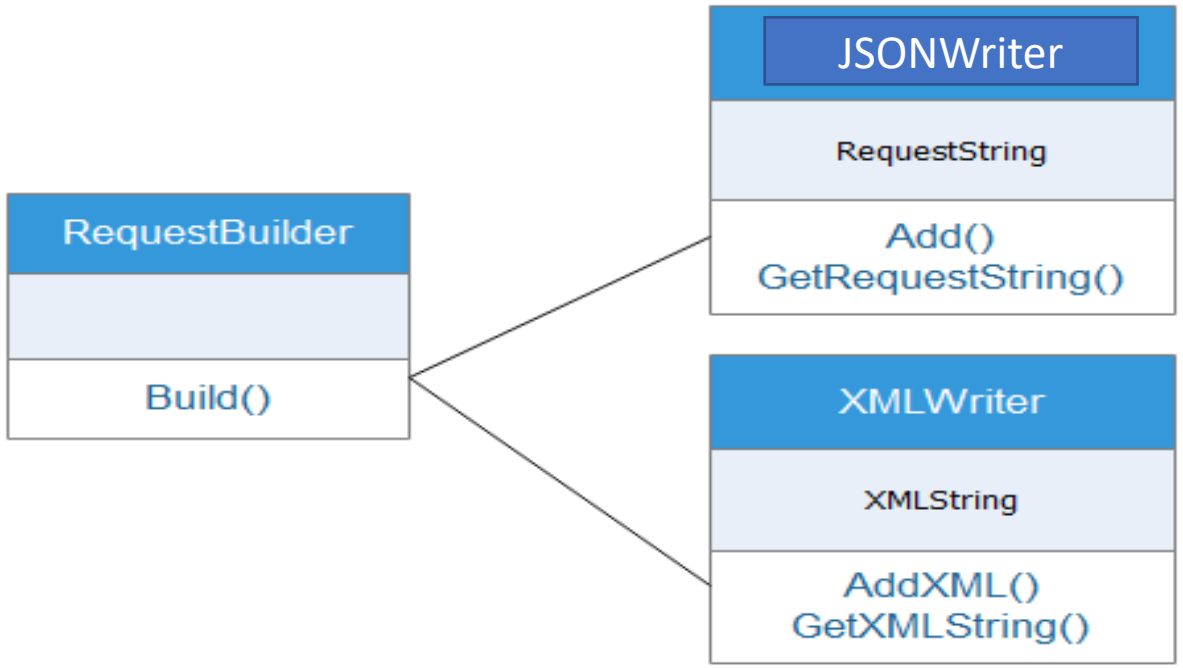




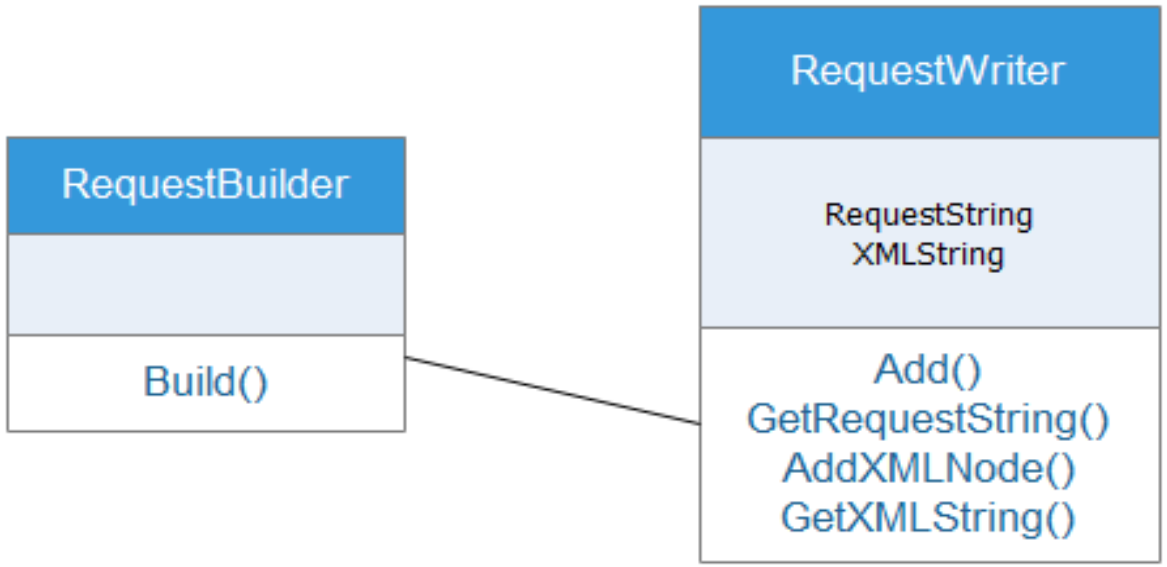
Add Method have multiple Responsibilities

ERROR Prone Design

Proposed Design 1



Proposed Design 2

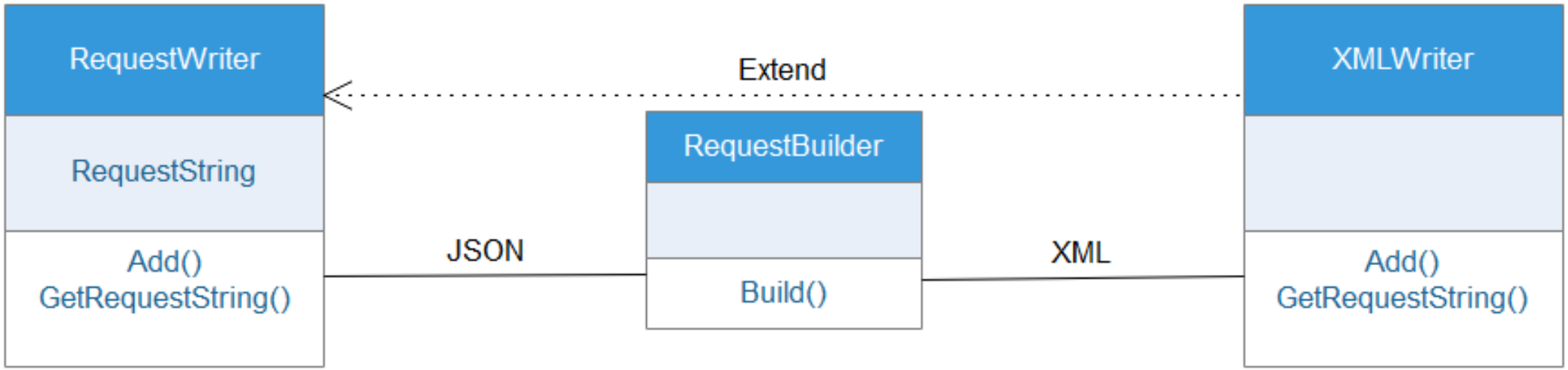


Open/Closed Principle

In programming, the open/closed principle states that software entities(Class, Methods) should be open for extension,
but closed for modification.

Liskov Substitution Principle

In programming, objects should be replaceable with instances of their subtypes without altering the correctness of that program.




```
Class Test.RequestBuilder Extends %RegisteredObject
{
/// we can pass object of XMLWriter / RequestWriter
Method Build(objWriter As RequestWriter)
{
        do objWriter.Add("Key1","Value1")
        do objWriter.Add("Key2","Value2")
        Set Request = objWriter.GetRequestString()
……
```

But there are still some stuffs, which are bothering me:

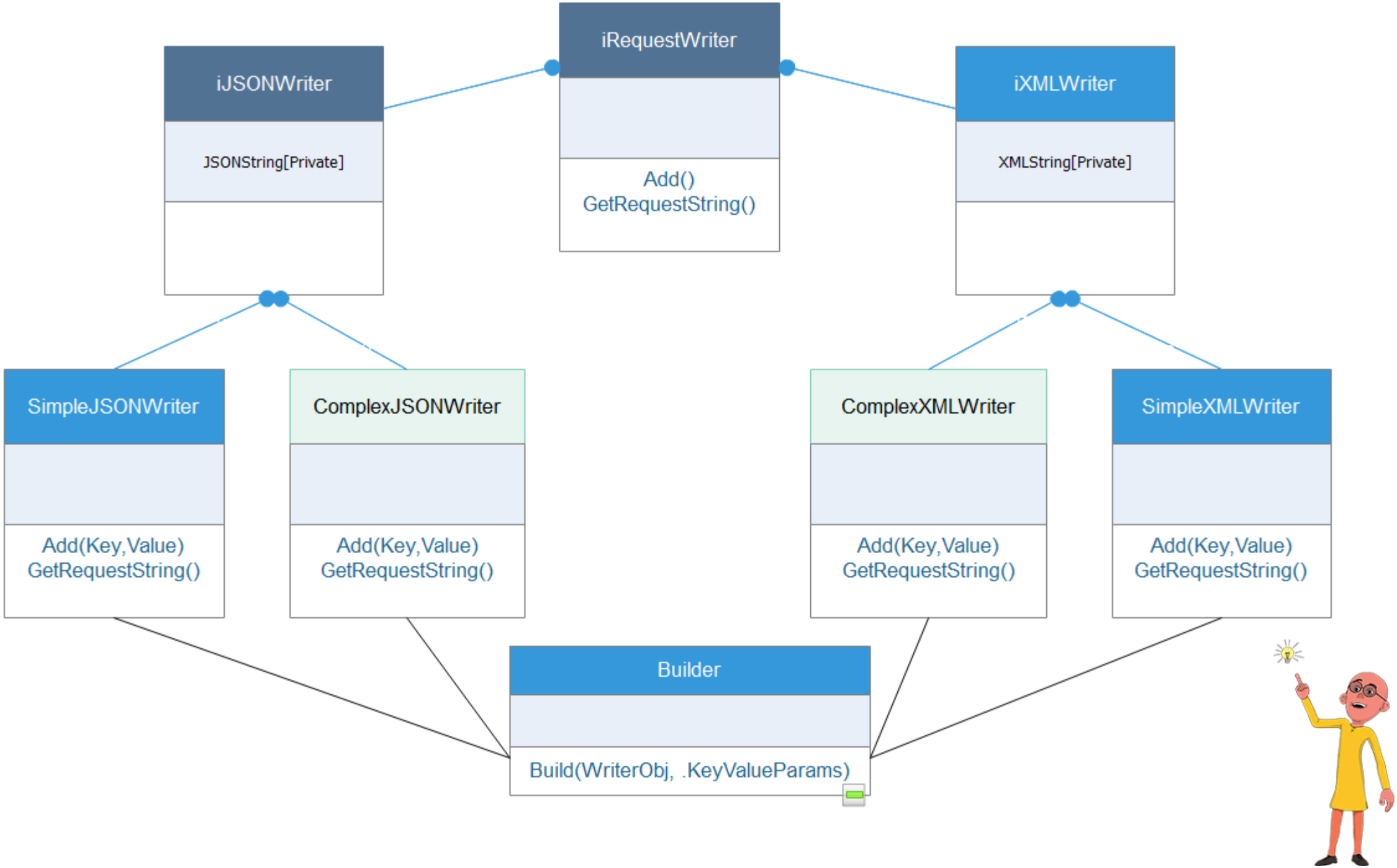
- Why RequestWriter, it should be JSONWriter.
- Currently, JSONWriter only supports Simple JSON having KEY - Value. What will happen if tomorrow, the request message gets complex like this.
  If we have to keep employee Children Details or Address, which might be Array of Object or Object embedded into JSON.
- For now, as requirement is only for Simple Structure, same can apply in XML Writer.
- WHY XMLWriter is extending RequestWriter/JSONWriter. There should be a generalized definition.

```
{
   “name” : “dsf”,
   “salary” : “XXXXXXX”,
    “age” : “XX”,
     “children”:
      [
         {
            “name “ : “sdf”,
         },
         {
            “name “ : “hgf”,
         }
      ],
     “homeAddress” :
          {
             “addressLine1” : “44 street”,
             “addressLine2” : “Zootopia”
          }
}
```

Interface Segregation Principle

Clients should not be forced to implement interfaces they don't use. Instead of one fat interface, many small interfaces are preferred based on groups of methods, each one serving one submodule.

In Cache Objects, we don’t have Interface Class, but we do have ABSTRACT class. So we can use ABSTRACT class as Interface class.


iRequestWriter
Add()
GetRequestString()
iJSONWriter
JSONString[Private]
iXMLWriter
XMLString[Private]
SimpleJSONWriter
Add(Key,Value)
GetRequestString()
ComplexJSONWriter
Add(Key,Value)
GetRequestString()
ComplexXMLWriter
Add(Key,Value)
GetRequestString()
SimpleXMLWriter
Add(Key,Value)
GetRequestString()
Builder
Build(WriterObj, .KeyValueParams)

```
{
  “name” : “dsf”,
  “salary” : “XXXXXXX”,
  “age” : “XX”,
  “children”:
  [
    {
      “name “ : “sdf”,
    },
    {
      “name “ : “hgf”,
    }
  ],
  “homeAddress” :
    {
      “addressLine1” : “44 street”,
      “addressLine2” : “Zootopia”
    }
}
```

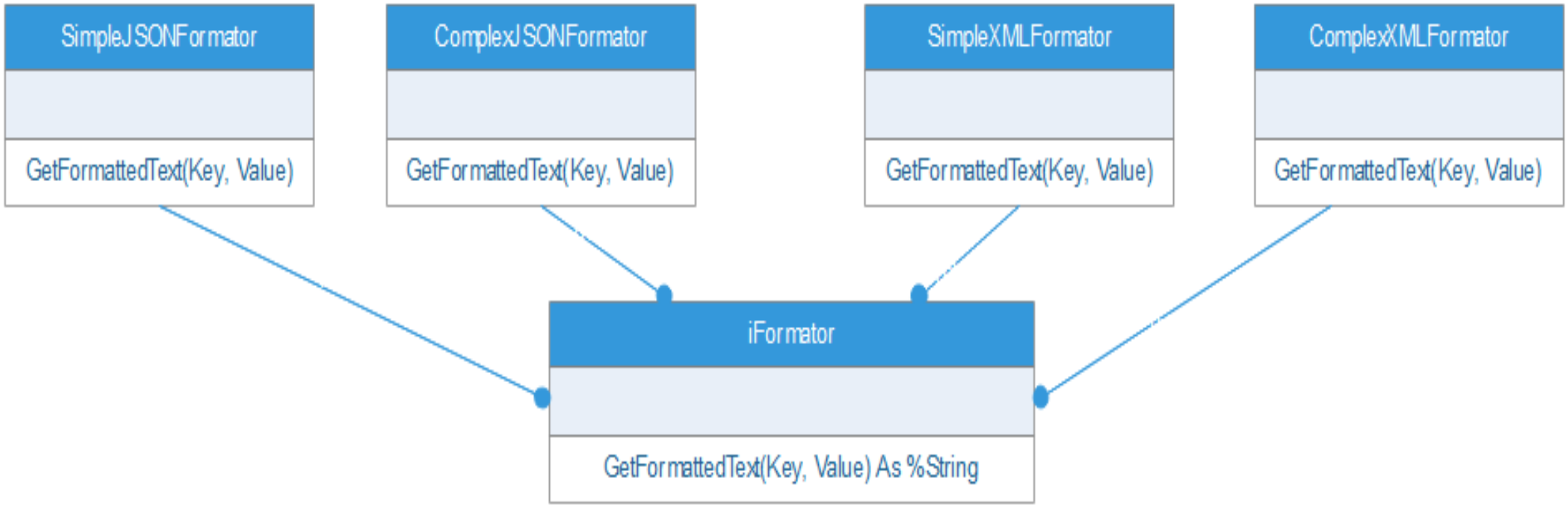
SIMPLE

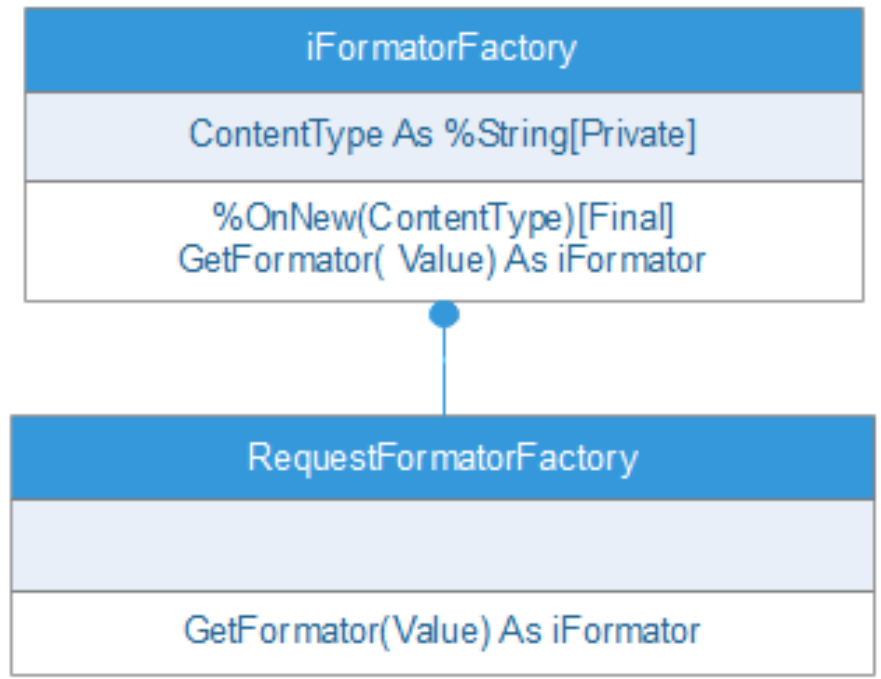
COMPLEX

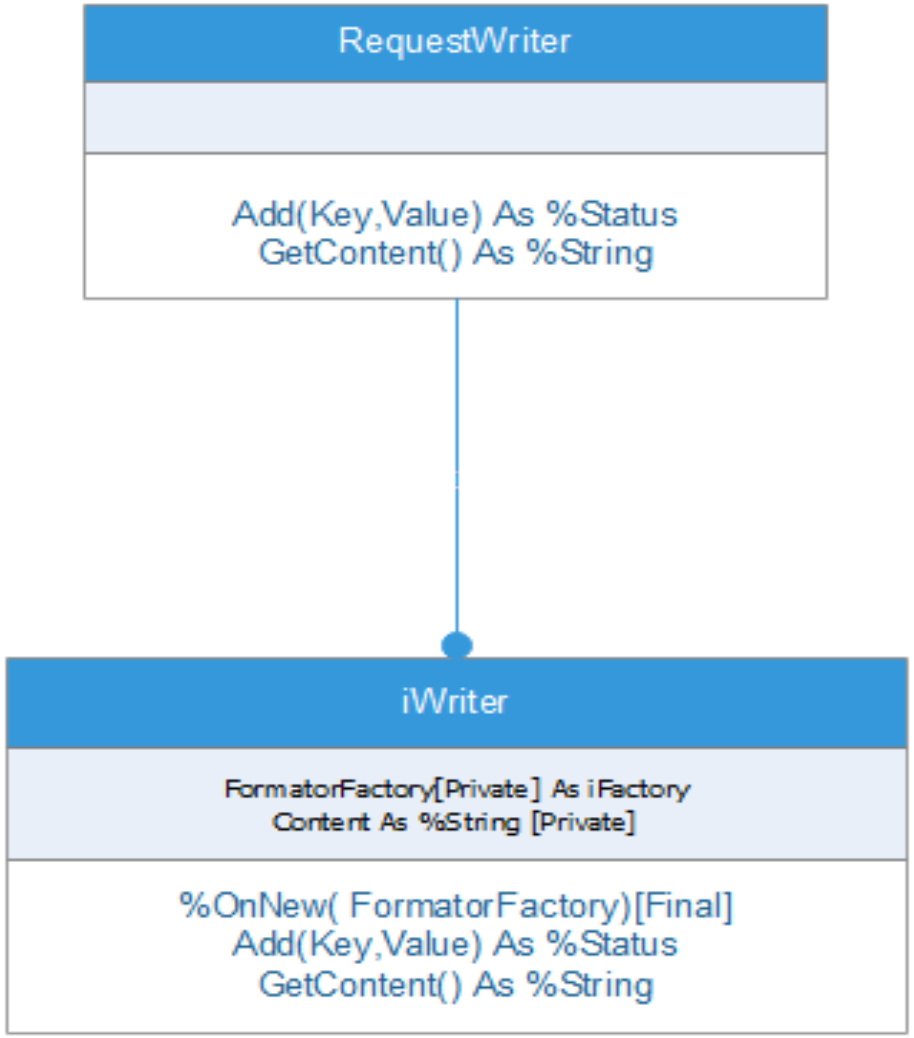
COMPLEX

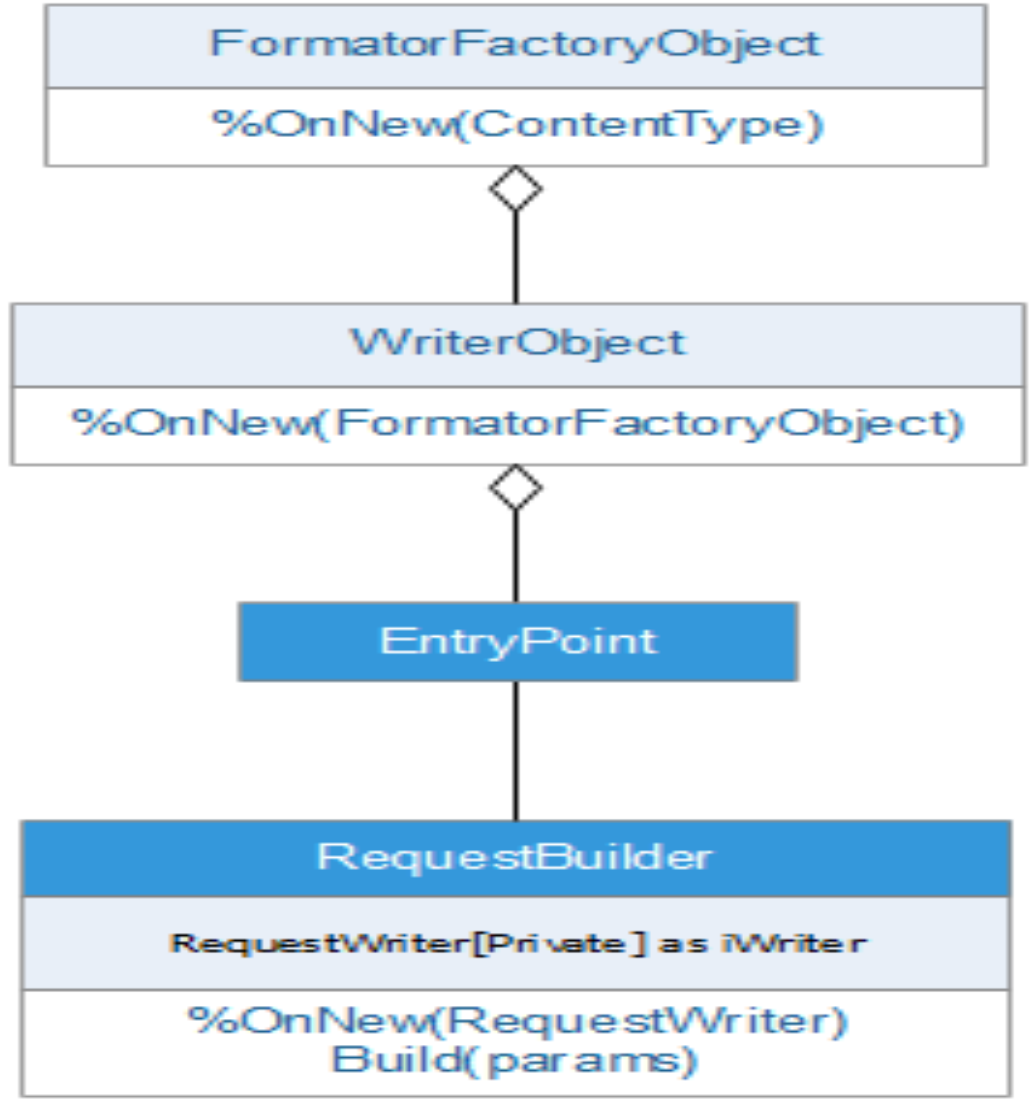
Dependency Inversion Principle

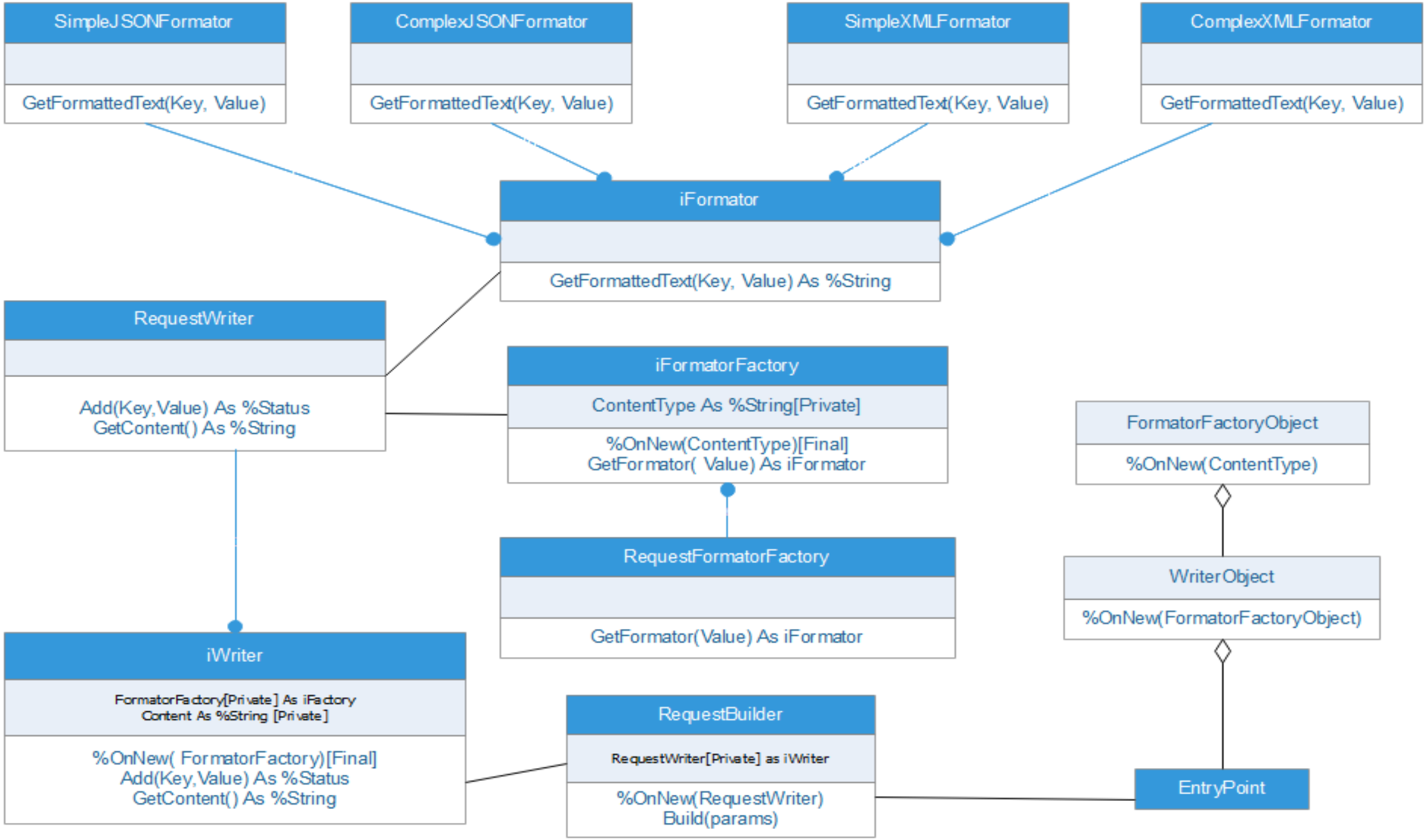
This principle is a way to decouple software modules.
To comply with this principle, we need to use Dependency Injection.
Dependency injection is used simply by ‘injecting’ any dependencies of a class through the class’ constructor as an input parameter.


SimpleJSONFormator
GetFormattedText(Key, Value)
ComplexJSONFormator
GetFormattedText(Key, Value)
SimpleXMLFormator
GetFormattedText(Key, Value)
ComplexXMLFormator
GetFormattedText(Key, Value)
iFormator
GetFormattedText(Key, Value) As %String


iFormatorFactory
ContentType As %String[Private]
%OnNew(ContentType)[Final]
GetFormator(Value) As iFormator
RequestFormatorFactory
GetFormator(Value) As iFormator


RequestWriter
Add(Key,Value) As %Status
GetContent() As %String
iWriter
FormatorFactory[Private] As iFactory
Content As %String [Private]
%OnNew(FormatorFactory)[Final]
Add(Key,Value) As %Status
GetContent() As %String


FormatorFactoryObject
%OnNew(ContentType)
WriterObject
%OnNew(FormatorFactoryObject)
EntryPoint
RequestBuilder
RequestWriter[Private] as iWriter
%OnNew(RequestWriter)
Build(params)


SimpleJSONFormator
GetFormattedText(Key, Value)
ComplexJSONFormator
GetFormattedText(Key, Value)
SimpleXMLFormator
GetFormattedText(Key, Value)
ComplexXMLFormator
GetFormattedText(Key, Value)
iFormator
GetFormattedText(Key, Value) As %String
RequestWriter
Add(Key,Value) As %Status
GetContent() As %String
iFormatorFactory
ContentType As %String[Private]
%OnNew(ContentType)[Final]
GetFormator(Value) As iFormator
RequestFormatorFactory
GetFormator(Value) As iFormator
FormatorFactoryObject
%OnNew(ContentType)
WriterObject
%OnNew(FormatorFactoryObject)
iWriter
FormatorFactory[Private] As iFactory
Content As %String [Private]
%OnNew(FormatorFactory)[Final]
Add(Key,Value) As %Status
GetContent() As %String
RequestBuilder
RequestWriter[Private] as iWriter
%OnNew(RequestWriter)
Build(params)
EntryPoint