```
 1: module ThothExamples
 2:
 3: open Fable.Core.JsInterop // !^
 4: open Fetch
 5: open Thoth.Json
 6: open Fable.Core
 7: open Thoth.Json
 8:
 9:
10:
11: module Print =
12:     let elementId = "elmish-app"
13:     let elem = Browser.Dom.document.getElementById(elementId)
14:     elem.setAttribute("style", "color:gray; margin:1rem; display: block;font-family: monospace;white-space: pre-wrap;"; )
15:
16:     let p input =
17:         let x = input
18:         let showElement = Browser.Dom.document.createElement("li")
19:         showElement.innerHTML <- sprintf "%A\n- - - - - - - - - - - - - - - - - - - -" x
20:         Browser.Dom.document.getElementById(elementId).appendChild showElement |> ignore
21:
22: //------------------------------
23:
24: module Random =
25:     let rand = System.Random()
26:     let int n = rand.Next(n)
27:     let float n = n * rand.NextDouble()
28:     let string n = System.String(Array.init n (fun _ -> char (rand.Next(97,123))))
29:     let bool = rand.NextDouble() >= 0.5
30:
31: //---------------------
32: // https://github.com/MangelMaxime/Thoth/issues/72
33: module DU001 =
34:     Print.p ("****************** DU001 ******************")
35:
36:     type Foo =
37:         { Name : string }
38:
39:         static member Decoder =
40:             Decode.object (fun get ->
41:                 { Name = get.Required.Field "name" Decode.string }
42:             )
43:         static member Encoder (x : Foo) =
44:             Encode.object [
45:                 "name", Encode.string x.Name
46:             ]
```

```
47:
48:     type Bar =
49:         { Value : int }
50:
51:         static member Decoder =
52:             Decode.object (fun get ->
53:                 { Value = get.Required.Field "value" Decode.int }
54:             )
55:         static member Encoder (x : Bar) =
56:             Encode.object [
57:                 "value", Encode.int x.Value
58:             ]
59:
60:     type MyDUWrapper =
61:         | FooWrapper of List<Foo>
62:         | BarWrapper of List<Bar>
63:
64:         static member Decoder =
65:             Decode.object (fun get ->
66:                 let typeDU = get.Required.Raw (Decode.field "type" Decode.string)
67:                 match typeDU with
68:                 | "FooWrapper" ->
69:                     get.Required.Field "values" (Decode.list Foo.Decoder)
70:                     |> FooWrapper
71:                 | "BarWrapper" ->
72:                     get.Required.Field "values" (Decode.list Bar.Decoder)
73:                     |> BarWrapper
74:                 | unknown ->
75:                     failwithf " %s isn't a valid type for MyDUWrapper" unknown
76:             )
77:         // NOT WORKING!
78:         // static member Decoder1 =
79:         //     Decode.oneOf [
80:         //         Decode.field "FooWrapper" (Decode.list |> Decode.map FooWrapper)
81:         //         Decode.field "BarWrapper" (Decode.list Bar.Decoder)
82:         //     ]
83:
84:
85:         static member Encoder (x : MyDUWrapper ) =
86:             match x with
87:             | FooWrapper fooValues ->
88:                 Encode.object [
89:                     "type", Encode.string "FooWrapper"
90:                     "values", fooValues |> List.map Foo.Encoder |> Encode.list
91:                 ]
92:             | BarWrapper barValues ->
```

```
 93:                 Encode.object [
 94:                     "type", Encode.string "BarWrapper"
 95:                     "values", barValues |> List.map Bar.Encoder |> Encode.list
 96:                 ]
 97:
 98:
 99:     let fooList =
100:         [
101:             { Name = "Maxime" }
102:             { Name = "Robert" }
103:             { Name = "Herve" }
104:         ]
105:
106:     let barList =
107:         [
108:             { Value = 27 }
109:             { Value = 17 }
110:             { Value = 227 }
111:             { Value = 257 }
112:         ]
113:
114:     let fooListJson =
115:         fooList
116:         |> List.map Foo.Encoder
117:         |> Encode.list
118:         |> Encode.toString 4
119:
120:     let barListJson =
121:         barList
122:         |> List.map Bar.Encoder
123:         |> Encode.list
124:         |> Encode.toString 4
125:
126:
127:     let myDUWrapperFooJson =
128:         FooWrapper fooList
129:         |> MyDUWrapper.Encoder
130:         |> Encode.toString 4
131:
132:     let myDUWrapperBarJson =
133:         BarWrapper barList
134:         |> MyDUWrapper.Encoder
135:         |> Encode.toString 4
136:
137:
138:     match Decode.fromString MyDUWrapper.Decoder myDUWrapperFooJson with
```

```
139:     | Ok values ->
140:         values |> Print.p
141:     | Error msg ->
142:         msg |> Print.p
143:
144:
145:     match Decode.fromString MyDUWrapper.Decoder myDUWrapperBarJson with
146:     | Ok values ->
147:         values |> Print.p
148:     | Error msg ->
149:         msg |> Print.p
150:
151: //----------------------
152: // https://github.com/MangelMaxime/Thoth/issues/79
153: module DU002 =
154:     Print.p ("****************** DU002 ******************")
155:
156:     type YingYang =
157:         | Ying of int
158:         | Yang of string
159:         static member Encoder = function
160:             | Ying x -> Encode.object [ "Ying", Encode.int x ]
161:             | Yang x -> Encode.object [ "Yang", Encode.string x ]
162:         static member Decoder =
163:             Decode.oneOf [
164:                 Decode.field "Ying" Decode.int |> Decode.map Ying
165:                 Decode.field "Yang" Decode.string |> Decode.map Yang
166:             ]
167:
168:     type RecordWithOption =
169:         { yingyang : YingYang option }
170:         static member Encoder x =
171:             Encode.object [ "yingyang", Encode.option YingYang.Encoder x.yingyang ]
172:         static member Decoder =
173:             Decode.object (fun get -> { yingyang = get.Optional.Field "yingyang" YingYang.Decoder } )
174:
175:     let json = """{"yingyang":null}"""
176:
177:     let expected = Ok({ yingyang = None})
178:
179:     let actual = Decode.fromString RecordWithOption.Decoder json
180:
181:     expected |> Print.p
182:     actual |> Print.p
183:
184: //-------
```

```
185:
186: // https://github.com/MangelMaxime/Thoth/issues/51
187: module Test =
188:
189:     open Thoth.Json
190:     Print.p ("****************** MODULE TEST ******************")
191:
192:     let log = Print.p
193:
194:     type Person =
195:         { Firstname : string
196:           Age : int option }
197:
198:         static member DecoderWithObject =
199:             Decode.object (fun get ->
200:                 { Firstname = get.Required.Field "firstname" Decode.string
201:                   Age = get.Optional.Field "age" Decode.int }
202:             )
203:
204:         static member DecoderWithMap2 =
205:             Decode.map2 (fun firstname age ->
206:                 { Firstname = firstname
207:                   Age = age } : Person )
208:                 (Decode.field "firstname" Decode.string)
209:                 (Decode.oneOf [ Decode.field "age" (Decode.option Decode.int); Decode.succeed None])
210:
211:
212:     let run () =
213:
214:         let jsonFull =
215:             """
216: {
217:     "firstname": "Maxime",
218:     "age": 26
219: }
220:             """
221:
222:         let jsonPartial =
223:             """
224: {
225:     "firstname": "Maxime"
226: }
227:             """
228:         log "Person.DecoderWithObject jsonFull"
229:         Decode.fromString Person.DecoderWithObject jsonFull
230:         |> log
```

```
231:
232:         log "Person.DecoderWithMap2 jsonFull"
233:         Decode.fromString Person.DecoderWithMap2 jsonFull
234:         |> log
235:
236:         log "Person.DecoderWithObject jsonPartial"
237:         Decode.fromString Person.DecoderWithObject jsonPartial
238:         |> log
239:
240:         log "Person.DecoderWithMap2 jsonPartial"
241:         Decode.fromString Person.DecoderWithMap2 jsonPartial
242:         |> log
243:
244: Test.run()
245:
246: //--------
247:
248: module Test2 =
249:     Print.p ("****************** Test 2 ******************")
250:
251:     open Thoth.Json
252:
253:     type Shape =
254:         | Circle of radius: int
255:         | Square of width: int * height: int
256:
257:         static member DecoderCircle =
258:             Decode.field "radius" ( Decode.int
259:                                     |> Decode.map Circle )
260:
261:         static member DecoderSquare =
262:             Decode.field "square" ( Decode.tuple2
263:                                         Decode.int
264:                                         Decode.int
265:                                     |> Decode.map Square )
266:
267:
268:     type MyObj =
269:         { Enabled: bool
270:           Shape: Shape }
271:
272:         static member Decoder =
273:             let commonDecoder shapeDecoder =
274:                 Decode.map2
275:                     (fun enabled shape ->
276:                         { Enabled = enabled
```

```
277:                           Shape = shape } )
278:                     (Decode.field "enabled" Decode.bool)
279:                     shapeDecoder
280:
281:             // We first need to detect the shape without moving further in the path
282:             // By using, ‘field‘ we can stay at the same position in the decoded object
283:             Decode.field "shape" Decode.string
284:             |> Decode.andThen (
285:                 function
286:                 | "circle" ->
287:                     commonDecoder Shape.DecoderCircle
288:
289:                 | "square" ->
290:                     commonDecoder Shape.DecoderSquare
291:
292:                 | unknown ->
293:                     sprintf "‘%s‘ is an unknown shape" unknown
294:                     |> Decode.fail
295:             )
296:
297:     let circleJson =
298:         """
299: {
300:     "enabled": true,
301:         "shape": "circle",
302:     "radius": 20
303: }
304:         """
305:
306:     let run () =
307:         Decode.fromString MyObj.Decoder circleJson
308:         |> Print.p
309:
310: Test2.run()
311:
```