DYALOG

Elsinore 2023

John has some generic problems<sad,face>
What can he do about them?

John Daintree

DYALOG

Elsinore 2023

John has some generic problems<sad,face>
What can he do about them?

John Daintree

DYALOG

Elsinore 2023

problems<sad,face>

John Daintree

DYALOG

Elsinore 2023

problems<sad,face>

What are "generics", and why do we care?

John Daintree

The problem with generics

```
using (var prod = new ProducerBuilder<string, byte[]>(_config).Build()) {
    for (int i=0; i<10; ++i) {
        prod.Produce(topic, new Message<string, byte[]> { Key = ..., Value = ... },
            (deliveryReport) => {
                if (deliveryReport.Error.Code != ErrorCode.NoError) {
                    Console.WriteLine($"{deliveryReport.Error.Reason}");
                } else {
                    // Message delivered
```

The problem with generics

```
string connectionString = "mongodb://grs3550";
var client = new MongoClient(connectionString);
var db = client.GetDatabase("sample_mflix");

var collection = db.GetCollection<BsonDocument>("movies");
var filter = Builders<BsonDocument>.Filter.Eq<string>("title", "Runaway");
var document = collection.FindSync<BsonDocument>(filter).First();
Console.WriteLine(document);
```

What are generics?

In statically typed languages (such as C++ and Java), the term *generic functions* refers to a mechanism for *compile-time polymorphism* (static dispatch), specifically parametric polymorphism. These are functions defined with TypeParameters, intended to be resolved with compile time type information. The compiler uses these types to instantiate suitable versions, resolving any function overloading appropriately.

https://en.wikipedia.org/wiki/Generic_function

What are (.NET) generics?

Generics are classes, structures, interfaces, and methods that have placeholders (type parameters) for one or more of the types that they store or use. A generic collection class might use a type parameter as a placeholder for the type of objects that it stores. The type parameters appear as the types of its fields and the parameter types of its methods. A generic method might use its type parameter as the type of its return value or as the type of one of its formal parameters.

https://learn.microsoft.com/en-us/dotnet/standard/generics

Generic Functions

int	MyFunction	(int arg1, int arg2)
double	MyFunction	(double arg1, double arg2)
char	MyFunction	(char arg1, char arg2)
object	MyFunction	(object arg1, object arg2)
Object	MyFunction2	(object[] args)

T1	MyFunction< T1 >	(T1 arg1, T1 arg2)
T	MyFunction2<T>	(T[])

Generic Functions

```
T1 MyFunction<T1> ( T1 arg1, T1 arg2 )
int result = MyFunction<int>(1,2);
int result = MyFunction<int>("1",2);
string result = MyFunction<string>("1","2");

T MyFunction2<T>(T[])
int result = MyFunction2<int>(new int[] {1,2,3});
int result = MyFunction2<int>(new string[] {"1","2","3"});
string result = MyFunction2<string>(new string[] {"1","2","3"});
```

Generic types

The problem with generics

```
using (var prod = new ProducerBuilder<string, byte[]>(_config).Build()) {
    for (int i=0; i<10; ++i) {
        prod.Produce(topic, new Message<string, byte[]> { Key = ..., Value = ... },
            (deliveryReport) => {
                if (deliveryReport.Error.Code != ErrorCode.NoError) {
                    Console.WriteLine($"{deliveryReport.Error.Reason}");
                } else {
                    // Message delivered
```

Generic types

```
string connectionString = "mongodb://grs3550";
var client = new MongoClient(connectionString);
var db = client.GetDatabase("sample_mflix");

var collection = db.GetCollection<BsonDocument>("movies");
var filter = Builders<BsonDocument>.Filter.Eq<string>("title", "Runaway");
var document = collection.FindSync<BsonDocument>(filter).First();
Console.WriteLine(document);
```

Generic Types

```
public class Dictionary<TKey,TValue>
{
public void Add (TKey key, TValue value);
public TValue this[TKey key] { get; set; }
public bool ContainsKey (TKey key);
public bool ContainsValue (TValue value);
…
}
Dictionary d=new Dictionary<int,int>();
d.Add(1,2);
d.Add(1,"hello");
int value=d[1];
```

Why do we care?

```
ProducerBuilder<TKey,TValue>
Message<TKey,TValue>
Builders<BsonDocument>.Filter.Eq<string>("title", "Runaway")
```

```
public class List<T>
public class Dictionary<TKey,TValue>
```

Why do we care?

- They are useful
- We may use them "indirectly"
  - Dictionary<String,String> get_Settings();
  - set_Settings(Dictionary<String,String> settings);

DKaf.NET

- C# ideally placed for prototyping this
- Performant - still wraps the native library
- Disadvantage: no AIX
- Disadvantage: 'modern' C# (generics) means we can't quite use Confluent.Kafka directly

Generic Functions

```
string connectionString = "mongodb://grs
var client = new MongoClient(connection$
var db = client.GetDatabase("sample_mfl

var collection = db.GetCollection<BsonDo
var filter = Builders<BsonDocument>.Filt
var document = collection.Find(filter).F
Console.WriteLine(document);
```

```
mm←db NewMethod'GetCollection<BsonDocument>'
col←mm.Invoke db(,⊂'movies')
```

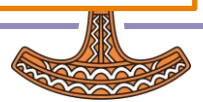
```
∇ dotnettype←type NewMethod methoddef;itemtypes;gt;m;its
;cutEndAt;cutFirst;split;trimEndAt;methodname;method
⍝ Requires ⎕using has '' as an item
cutEndAt←{(1-(⍵)⍳⍺)↑⍵}
cutFirst←{(¯1+⍵⍳⍺)↑⍵}
split←{1↓¨(⍺=⍺,⍵)⊂⍺,⍵}
trimEndAt←{(-(⍵)⍳⍺)↓⍵}
methodname←'<'cutFirst methoddef
itemtypes←','split'>'trimEndAt(1+≢methodname)↓methoddef

'No itemtypes'⎕SIGNAL((0=≢itemtypes)/90)

⍝gt←2017⌶generictype,'`',⍕≢itemtypes
gt←type.GetType
method←gt.GetMethod ⊂methodname
its←2017⌶¨itemtypes

:If ∨/m←⎕NULL≡¨its
      ((⍕m/itemtypes),'not found')⎕SIGNAL 90
:EndIf

:If gt≢⎕NULL
:AndIf method.IsGenericMethodDefinition
      dotnettype←method.MakeGenericMethod⊂its
:Else
      (generictype,' is not a generic type')⎕SIGNAL 90
:EndIf
∇
```



Generic Types

```
public class Producer
    public IProducer<
    private ProducerC

    public Producer(s
        _config = new
            Bootstrap
        };

        Prod = new ProducerBuilder<string, byte[]>(_config).Build();
    }
```

```
∇ Ctor (type arg)
    :Access public
    :Implements constructor
    ⍝ arg is either a charvec or …
    :If type=0
      producer←⎕NEW DyKa.StringProducer(⊂arg)
    :Else
      producer←⎕NEW DyKa.Producer(⊂arg)
    :End
∇
```

DKaf.NET

```
using (var prod = new ProducerBuild
    for (int i=0; i<10; ++i) {
        prod.Produce(topic, new Mes
            (deliveryReport) => {
                if (deliveryReport.
                    Console.WriteLi
                } else {
                    // Message delivered
```

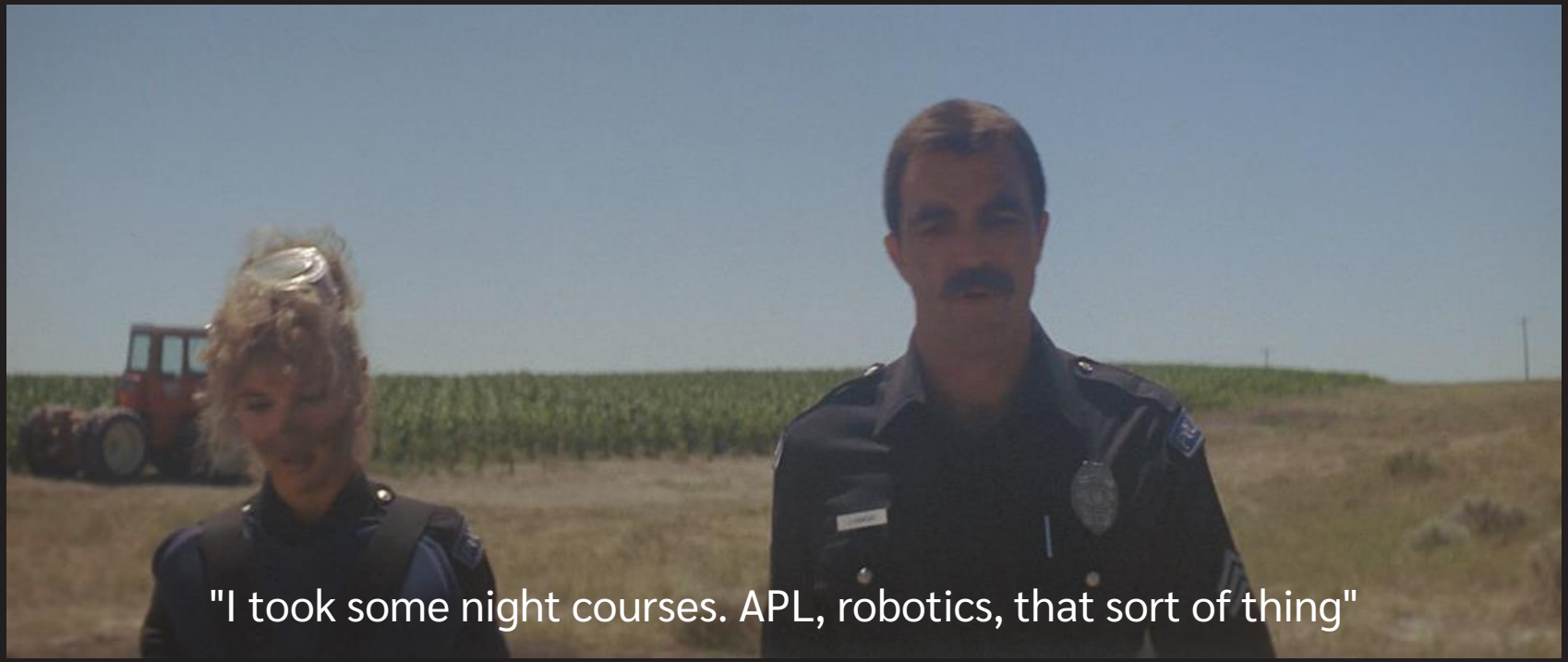
```
∇ genericObjType←genericType GenericOf itemTypes;gt;⎕USING;its
⍝ GenericType eg List etc
⍝ ItemTypes type of items
⎕USING←''
its←System.Type.GetType¨⊂¨,⊆itemTypes
gt←System.Type.GetType⊆genericType,'`',⍕≠its
:If gt≠⎕NULL
:AndIf gt.IsGenericTypeDefinition
      genericObjType←gt.MakeGenericType⊂its
:Else
      (genericType,' is not a generic type')⎕SIGNAL 90
:EndIf

∇
```

```
'Confluent.Kafka.ProducerBuilder' GenericOf 'System.String' 'System.Byte[]'
```

But we can do better

But we can do better


"I took some night courses. APL, robotics, that sort of thing"

Why do we care?

Other languages (e.g. C#) use generics extensively and increasingly
We need to be able to consume their public APIs
We should be able to provide generics to them

```
:Class oStats
:Using System
:Using System.Collections

∇ make args
:Signature ctor Object[]
:Access public
:Implements constructor
_array←args
∇

∇ r←min
:Signature Object←∇
:Access public

r←{⊃ω[⍋ω]}_array
∇
```

```
∇ r←max
:Signature Object←∇
:Access public

r←{⊃ω[⍒ω]}_array
∇

∇ r←unique
:Signature Object[]←∇
:Access public

r←∪_array
∇

∇ r←freq;tal
:Signature Hashtable←∇
:Access public
r←⎕NEW Hashtable
r.Add¨↓(⍳,≠⌸⊢)⌸_array
∇

:EndClass
```

```
:Class oStats
:Using System
:Using System.Collections

∇ make args
:Signature ctor Object[]
:Access public
:Implements constructor
_array←args
∇

∇ r←min
:Signature Object←∇
:Access public

r←{⊃ω[⍋ω]}_array
∇
```

```
∇ r←max
:Signature Object←∇
:Access public

r←{⊃ω[⍒ω]}_array
∇

∇ r←unique
:Signature Object[]←∇
:Access public

r←∪_array
∇

∇ r←freq;tal
:Signature Hashtable←∇
:Access public
r←⎕NEW Hashtable
r.Add¨↓(⍳,≠⌸⊢)⍸_array
∇

:EndClass
```

```
:Class gStats•T
:Using System
:Using
System.Collections.Generic

∇ make args
:Signature ctor T[]
:Access public
:Implements constructor
_array←args
∇

∇ r←min
:Signature T←∇
:Access public

r←{⊃ω[⍋ω]}_array
∇
```

```
∇ r←max
:Signature T←∇
:Access public

r←{⊃ω[⍒ω]}_array
∇

∇ r←unique
:Signature T[]←∇
:Access public

r←∪_array
∇

∇ r←freq;tal
:Signature Dictionary•T Int32←∇
:Access public

r←⎕NEW (Dictionary•T Int32)
r.Add ↓(⍳,≠⌸⊢)⍸_array
∇

:EndClass
```

Exporting Generics

DEMO

Generics

Questions and things I probably forgot.

Does it make sense to use generics in "APL only" cases?
"Extension methods"
TypeParameter inference

DYALOG

Elsinore 2023

John has some generic problems<sad,face>
What can he do about them?

John Daintree

- Fin.

```
public class Producer : IDisposable {
    public IProducer<string, byte[]>? Prod { get; private set; }
    private ProducerConfig _config;

    public Proc
        _config
            Boo
        };

        Prod =
    }
```

```
∇ Ctor (type arg)
  :Access public
  :Implements constructor
  ⍝ arg is either a charvec or …
  :If type=0
    producer ← ⎕NEW DyKa.StringProducer(⊂arg)
  :Else
    producer ← ⎕NEW DyKa.Producer(⊂arg)
  :End
∇
```

Generic Functions

T1 MyFunction<T1,T2>(T1 arg1, T2 arg2)
int result=MyFunction<int,string>(10,"hello");
String result=MyFunction<string,string>("hello","world");

Generic Functions

```
int Find<T1>(T1 arg1, T1[] in)
int Find<target>(target arg1, target[] in)
int index=Find<int>(10          ,new int[] {10,11,12});
int index=Find<string>("john",new string[] {"hello","johnd"});
int index=Find<string>(10      ,new string[] {"hello","johnd"});
```

Generic Functions

```
var client = new MongoClient(connectionString);
var db = client.GetDatabase("sample_mflix");
var collecton=db.GetCollection<BsonDocument>("movies");
var filter = Builders<BsonDocument>.Filter.Eq<string>("title", "Runaway");
var document = collection.Find(filter).First();
```

Generic Functions

Generic Functions

int problems<sad,face>(sad john, face saving)

These things exist, but we can’t call them from Dyalog APL.
… until now.

Generic Classes

Defining Generics in Dyalog