C#6

A cleaner code

LAB

Rui Carvalho
@rhwy

Maxime Sanglan
@__MaxS__

next >

Agenda

let’s spend some time together

< previous

next >

- Check if system is running
- Test project in place
- Concepts presentation
- A few exercices
- A little Kata to finish ;-)

What do I Need?

** not the usual .Net boiler plate **

previous

next

Install Mac/Linux

01

```
1.Install .Net engine version manager
_____________________________________
curl -sSL https://dist.asp.net/dnvm/
dnvminstall.sh | sh && source ~/.dnx/dnvm/
dnvm.sh
```

< previous

next >

Install Mac/Linux

02

```
2.Install latest .Net engine
_____________________________
dnvm install latest

dnvm install 1.0.0-rc1-update1
```

Install Mac/Linux

03

```
3.Verify
________
dnu --version
dru —-help
```

previous

next

Install Mac/Linux

04

```
1.Install .Net engine version manager
_____________________________________
curl -sSL https://dist.asp.net/dnvm/dnvminstall.sh |
sh && source ~/.dnx/dnvm/dnvm.sh

2.Install latest .Net engine
____________________________
dnvm install latest

3.Verify
________
dnu --version
```

Setup workspace

** we should always start with a test project ;-) **

< previous

next >

Commands


```
#tooling
dnu  --help

#execution
#(just like `node myapp` or `java myapp`
dnx {myapp}

#default
dnx run
```

Minimal

project.json

```
{
 "frameworks": {
    "dnx451": {}
  }
}
```

app.cs

```
public class Program
{
    public void Main(params string[] args)
    {
        System.Console.WriteLine("hello Mix-IT");
    }
}
```

Testing

project.json

```
{
 "dependencies": {
    "xunit": "2.1.0",
    "xunit.runner.dnx": "2.1.0-rc1-build204"
  },
  "commands": {
      "test":"xunit.runner.dnx"
  },
  "frameworks": {
    "dnx451": {}
  }
}
```

Testing

test.cs

```
using Xunit;

public class IceBreaker
{
    [Fact] public void
    my_first_test()
    {
        Assert.True(false);
    }
}
```

Better Testing?

project.json

```
"dependencies": {
    "xunit": "2.1.0",
    "xunit.runner.dnx": "2.1.0-rc1-build204",
    "nfluent": "1.3.1",
    "watchbird": "1.0.0-rc1-3"
  },
  "commands": {
    "test": "xunit.runner.dnx",
    "watch": "watchbird --dnx test"
  }
```

dnx watch

test.cs

```
using Xunit;using NFluent;

public class IceBreaker
{
    [Fact] public void my_first_test()
    {
        Assert.True(false);
        Check.That(MyList).Contains(1,2,3);
    }
}
```

C#6, What's New?

previous

next

Philosophy design

Clean up your code!

No New Big features
… but many small ones to
to improve your code

previous

next

Put C#6 In context

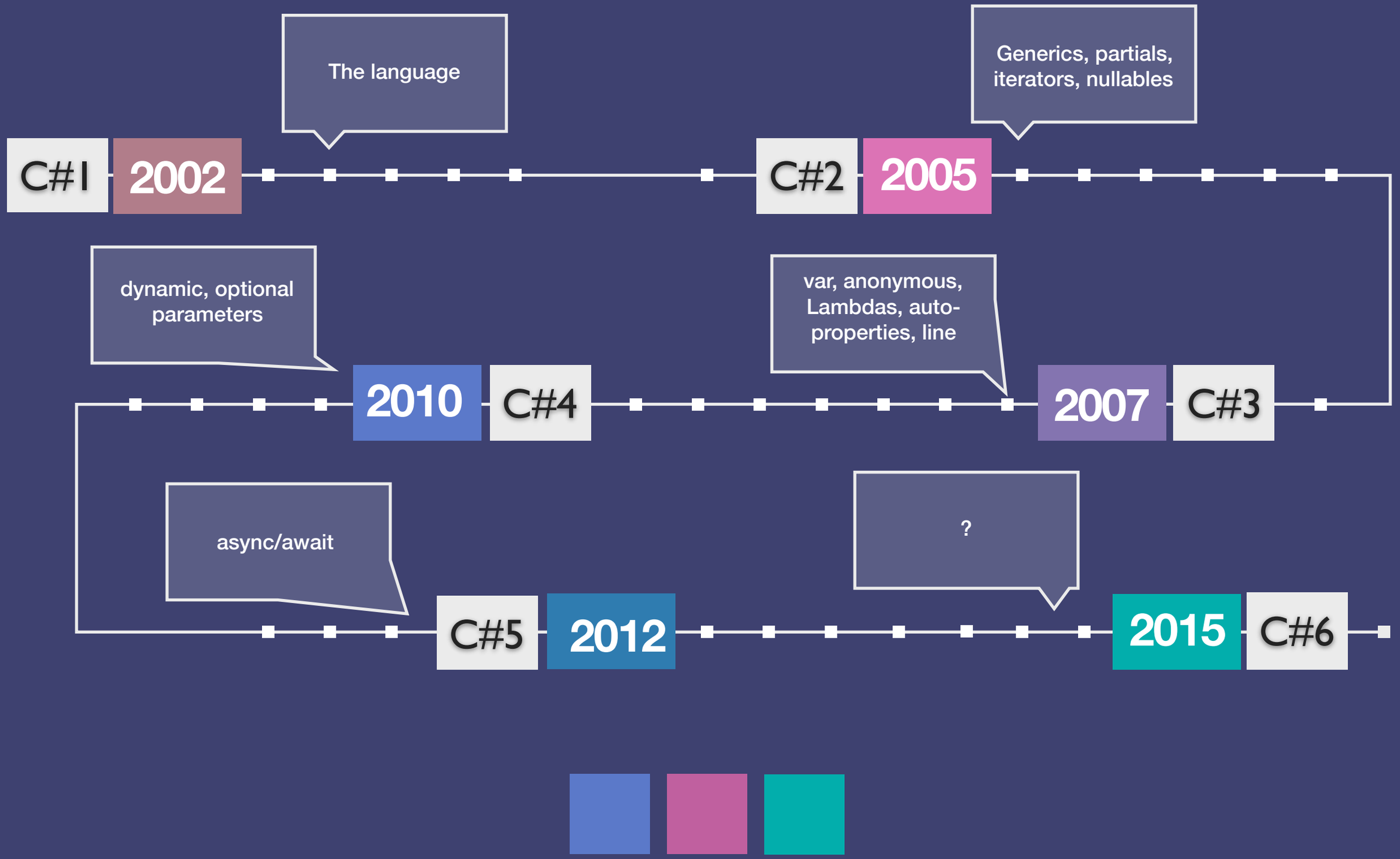
Clean up your code!

Let’s see how to **improve** our code with ***less boiler plate*** and ***more meanings***

previous

next

History



< previous

next >

Developers should write their code to fit in a slide

< previous

next >

News

01 Using Static classes

why bother with static class names?

02 String Interpolation

because string.format sucks

03 Getter only properties

because un-meaningful boiler plate

04 auto-properties initializers

because creating a backing field for properties only don't make sense

< previous

next >

Using Static

01

```
using System;

public class Code
{
  public void LogTime(string message)
  {
    Console.Write(DateTime.Now);
    Console.WriteLine(message);
  }
}
```

Using Static

01

```
using static System.Console;
using static System.DateTime;

public class Code
{
   public void Print(string message)
   {
      Write(Now);
      WriteLine(message);
   }
}
```

previous

next

Using Static

01

```
Console.Write(DateTime.Now);
```

From a very technical line

```
Write(Now);
```

You moved to a natural language reading friendly sentence

using static allows removal of technical boilerplate

String Interpolation

02

```
public void UserMessagesLog(
  string message, User user)
  {
      var messageToLog =
        string.Format("[{0}] {1} - {2}",
        DateTime.Now,
        user.Name,
        message);

      Console.WriteLine(messageToLog);
  }
}
```

String Interpolation

02

```
public void UserMessagesLog(
  string message, User user)
  {
      var messageToLog =
      $"[{DateTime.Now}] {User.Name} - {message}"
      Console.WriteLine(messageToLog);
    }
}
```

< previous

next >

String Interpolation

02

```
string.Format("[{0}] {1} - {2}",
  DateTime.Now,
  user.Name,
  message);
```

sounds good enough for small strings, But...

```
$"[{DateTime.Now}] {User.Name} - {message}"
```

we're **stupid**, when you ***read the code***, there will be a **context switch** to interpret the result of your string ***if the value is not in the place*** you wrote it!

< previous

next >

String Interpolation

02

```
public void UserMessagesLog(
  string message, User user)
  {
      var messageToLog =
      $"[{DateTime.Now}] {User.Name} - {message}"
      Console.WriteLine(messageToLog);
    }
}
```

Why waiting 15 years for that?

Getter only properties


```
public class Post
{
  private string title;
  public string Title
  {
    get{return title;}
  }
  public Post(string title)
  {
    this.title=title
  }
}
```

Getter only properties

03

```
public class Post
{
  public string Title { get;private set;}

  public Post(string title)
  {
    Title=title
  }
}
```

< previous

next >

Getter only properties

03

```
public class Post
{
  public string Title { get;}

  public Post(string title)
  {
    Title=title
  }
}
```

< previous

next >

AutoProperties initializers 04

```
public class Post
{
  private string id="new-post";
  public string Id
  {
    get{return id;}
  }
  public Post()
  {
  }
}
```

< previous

next >

AutoProperties initializers 04

```
public class Post
{
  public string Id {get;private set;}
  public Post()
  {
    Id="new-post";
  }
}
```

previous

next

AutoProperties initializers

04

```
public class Post
{
  public string Id {get;} = "new-post";
}
```

< previous

next >

News

05 Expression bodied properties

because one expression is enough

06 Expression bodied methods

too much braces for a one expression function

07 Index initializers

because initializer shortcuts are great

08 Null conditional operators

if null then null else arg…

< previous

next >

Expression bodied props 05

```
public class Post
{
  private DateTime created = DateTime.Now;
  public DateTime Created
  {
    get{return created;}
  }
  public TimeSpan Elapsed
  {
    get {
      return (DateTime.Now-Created);
    }
  }
}
```

< previous

next >

Expression bodied props 05

```
public class Post
{
  public DateTime Created {get;}
    = DateTime.Now;

  public TimeSpan Elapsed
      => (DateTime.Now-Created);
}
```

Expression bodied props

05

```
public class Post
{
  public DateTime Created {get;}
    = DateTime.Now;

  public DateTime Updated
    => DateTime;
}
```

Spot the difference

{get;} =	value is affected once for all, stored in an hidden backing field
=>	represents an expression, newly evaluated on each call

< previous

next >

Expression Bodied Methods 06

```
public class Post
{
  public string Title {get;set;}
  public string BuildSlug()
  {
    return Title.ToLower().Replace(" ", "-");
  }
}
```

< previous

next >

Expression Bodied Methods 06

```
public class Post
{
  public string Title {get;set;}
  public string BuildSlug()
  {
    return Title.ToLower().Replace(" ", "-");
  }
}
```

never been frustrated to have to write this **{return}** boilerplate since you use the **=>** construct with lambdas?

< previous

next >

Expression Bodied Methods 06

```
public class Post
{
  public string Title {get;set;}
  public string BuildSlug()
  => Title
      .ToLower()
      .Replace(" ", "-");
}
```

Index initializers


```
public class Author
{
  public Dictionary<string,string> Contacts
    {get;private set;};

  public Author()
  {
     Contacts = new Dictionary<string,string>();
     Contacts.Add("mail", "demo@rui.fr");
     Contacts.Add("twitter", "@rhwy");
     Contacts.Add("github", "@rhwy");
  }
}
```

Index initializers

07

```
public class Author
{
  public Dictionary<string,string> Contacts
    {get;}
    = new Dictionary<string,string>() {
        ["mail"]="demo@rui.fr",
        ["twitter"]="@rhwy",
        ["github"]="@rhwy"
    };
}
```

< previous

next >

Index initializers


```
public class Author
{
  public Dictionary<string,string> Contacts
    {get;}
    = new Dictionary<string,string>() {
        {"mail","demo@rui.fr"},
        {"twitter","@rhwy"},
        {"github","@rhwy"}
    };
}
```

Null Conditional Operators 08

```
public class Post
{
   //"rui carvalho"
   public string Author {get;private set;};
   //=>"Rui Carvalho"
   public string GetPrettyName()
   {
        if(Author!=null)
        {
           if(Author.Contains(" "))
           {
              string result;
              var items=Author.Split(" ");
              forEach(var word in items)
              {
                 result+=word.SubString(0,1).ToUpper()
                    +word.SubString(1).ToLower()
                    + " ";
              }
              return result.Trim();
           }
        }
        return Author;
   }
}
```

< previous

next >

Null Conditional Operators 08

```
public class Post
{
  //"rui carvalho"
  public string Author {get;private set;};
  //=>"Rui Carvalho"
public string GetPrettyName()
    => (string.Join("",
        Author?.Split(' ')
        .ToList()
        .Select( word=>
           word.Substring(0,1).ToUpper()
           +word.Substring(1).ToLower()
           + " ")
      )).Trim();
```

< previous

next >

Null Conditional Operators 08

```
public class Post
{
  //"rui carvalho"
  public string Author {get;private set;};
  //=>"Rui Carvalho"
public string GetPrettyName()
      => (string.Join("",
            Author?.Split(' ')
            .ToList()
            .Select( word=>
               word.Substring(0,1).ToUpper()
               +word.Substring(1).ToLower()
               + " ")
         )).Trim();
```

We have now one expression only but maybe not that clear or testable ?

< previous

next >

Null Conditional Operators 08

```
public string PrettifyWord (string word)
=>
   word.Substring(0,1).ToUpper()
   +word.Substring(1).ToLower()
   + " ";

public IEnumerable<string>
PrettifyTextIfExists(string phrase)
=> phrase?
   .Split(' ')
   .Select( PrettifyWord);

public string GetPrettyName()
=> string
   .Join("",PrettifyTextIfExists(Author))
   .Trim();
```

Refactoring !

We have now 3 small *independent functions* with *Names and meanings*

News

09 Exception filters

not new in .Net

10 nameof Operator

consistent refactorings

11 await in catch

who never complained about that?

12 and in finally

the last step

< previous

next >

News

09

Excepti

not new in .

catch

omplained about that?

10

nameo

consistent

finally

ep

these ones are just good improvements but not that important for our code readability

Exception Filters

09

```
try {
  //production code
}
catch (HttpException ex)
{
  if(ex.StatusCode==500)
    //do some specific crash scenario
  if(ex.StatusCode==400)
    //tell client that he's stupid
}
catch (Exception otherErrors)
{
  // ...
}
```

Exception Filters

09

```
try {
  //production code
}
catch (HttpException ex) when (ex.StatusCode == 500)
{
  //do some specific crash scenario
}
catch (HttpException ex) when (ex.StatusCode == 400)
{
  //tell the client he’s stupid
}
catch (Exception otherException)
{
  // ...
}
```

< previous

next >

nameof Operator


```
public string SomeMethod (string word)
{
  if(word == null)
    throw new Exception("word is null");
  return word;
}
```

nameof Operator


```
public string SomeMethod (string text)
{
  if(word == null)
    throw new Exception("word is null");
  return word;
}
```

arg, information lost!

Refactoring

C#3

< previous

next >

nameof Operator


```
public string SomeMethod (string word)
{
  if(word == null)
    throw new Exception(
      $"{nameof(word)} is null");
  return word;
}
```

parameter name is now pure code & support refactoring

C#6

< previous

next >

await in catch


```
try {
  return await something();
}
catch (HttpException ex)
{
  error = ex;
}
return await CrashLogger.ServerError(error);
```

< previous

next >

await in catch


```
try {
  await something();
}
catch (HttpException ex)
{
  await CrashLogger.ServerError(ex);
}
```

await in finally


```
try {
  return await Persistence.Query(query);
}
catch (PersistenceException ex)
{
  await CrashLogger.ServerError(ex);
}
finally
{
  await Persistence.Close();
}
```

To Finish

C#6 helps to be more functional & write cleaner code by removing lots of boilerplate!

refactoring to small functions, add new words to your app vocabulary

< previous

next >

Exercices

previous

next

Exercices

Constraint:

Refactor existing code to a more fluent version with C#6

Subject:

For each sample you have a simple class and an existing test to verify the behavior, refactoring should be safe ;-)

< previous

next >

Kata

previous

next

Kata

Constraint:

Write all with single lige methods

Subject:

Give a text representation of the elapsed time for a date within the predefined ranges:

more than 3 months, 2 months ago, 1 month ago, x days ago, x hours ago, x minutes ago

< previous

next >

Thanks

@rhwy & @__MaxS__

ncrafts.io

12-13 May 2016

< previous

github.com/rhwy

next >