Leveraging C# Extensibility Points

Part 2

Bart J.F. De Smet
bartde@outlook.com

pluralsight
hardcore dev and IT training

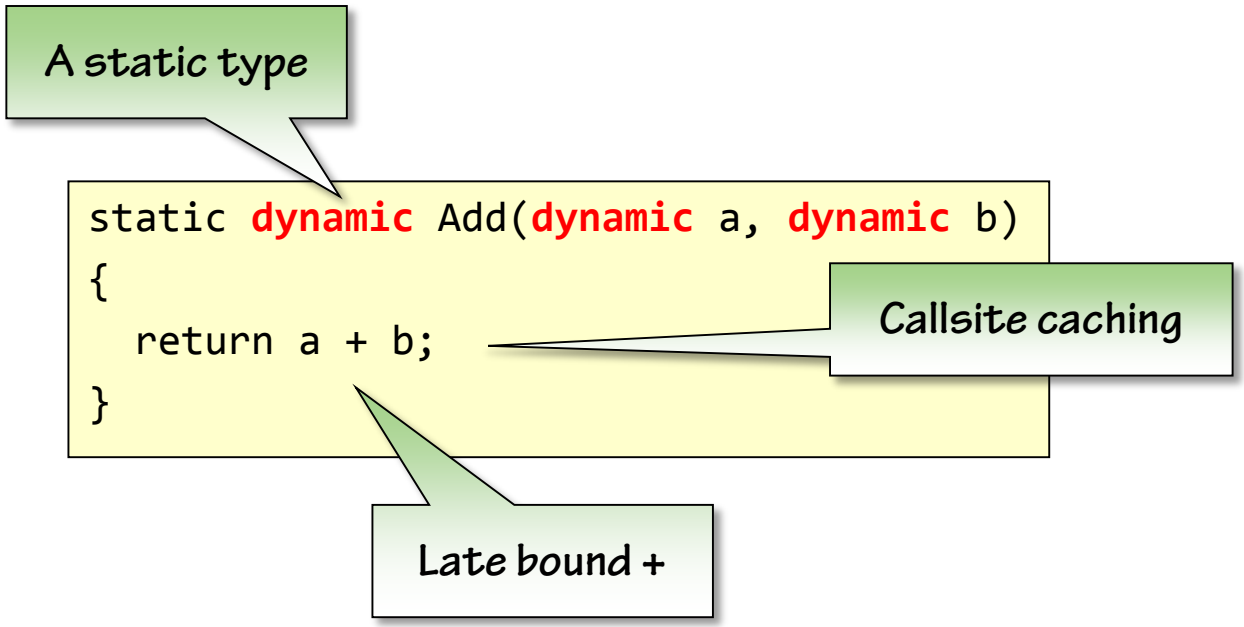
Dynamic Typing

- **The "dynamic" type in C# 4.0**
  - Any object can be assigned to "dynamic"
    - Really just "object" under the hood
    - Compiler tracks the (static) "dynamic" type
  - Operations on dynamically typed variables are **late bound**

- **Compilation**
  - Compiler emits dynamic call sites
  - Parameterized with Microsoft.CSharp.dll language binders

- **Dynamic Language Runtime (DLR)**
  - Language binder performs late binding
    - Using runtime type information (GetType)
    - Overload resolution, etc.
    - Emit expression trees
  - DLR compiles and caches call sites

Dynamic Typing

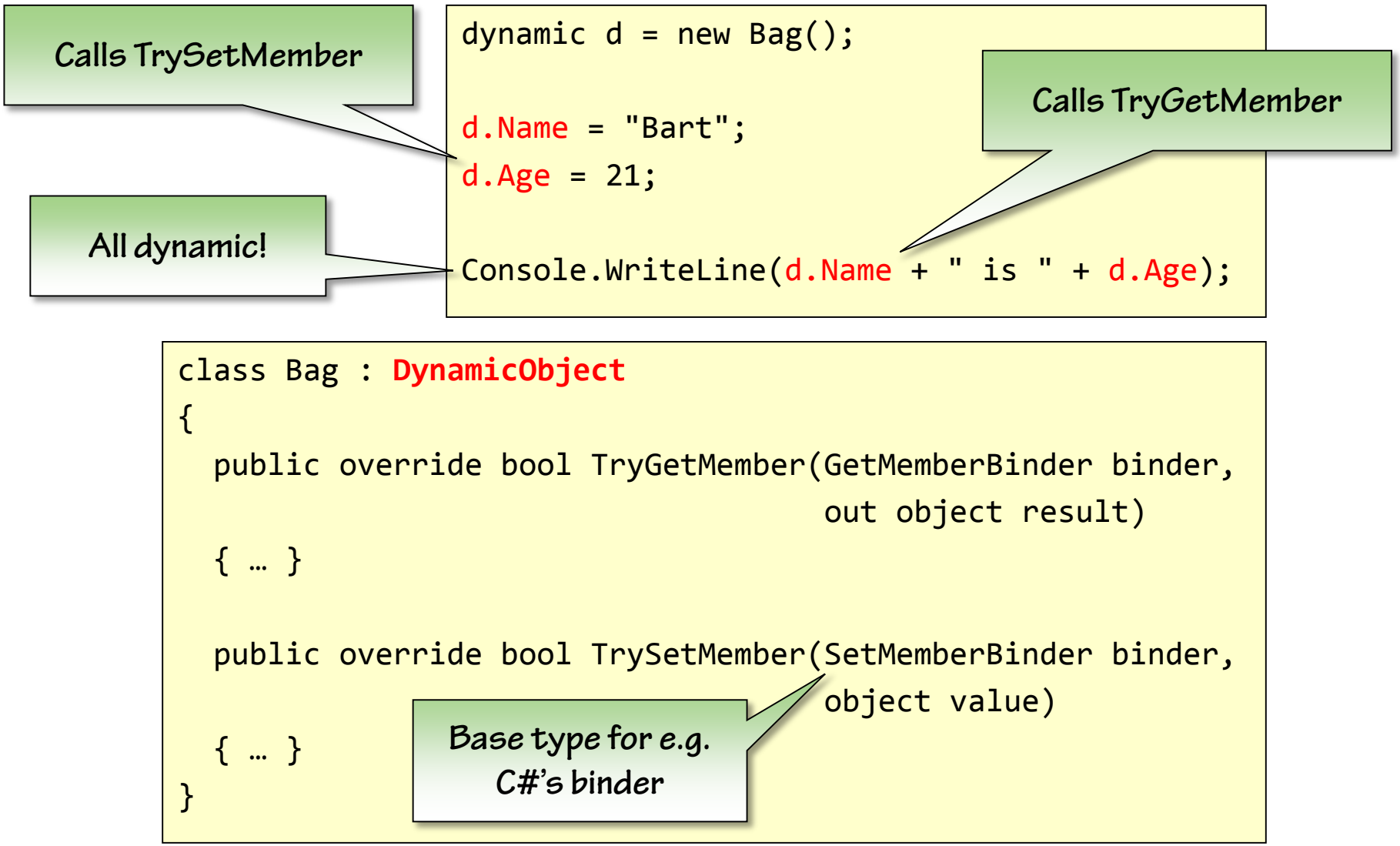



```
static void Main()
{
  dynamic x = Add(1, 2);
  dynamic d = Add(DateTime.Now, TimeSpan.Zero);
  dynamic s = Add("Hello, ", "Dynamic!");
  dynamic y = Add(3, 4);
}
```

Dynamic Typing

- **Leveraging the dynamic type**
  - Interop with dynamic languages, e.g. IronPython, JavaScript
  - Access to weakly typed data, e.g. XML, JSON
- **Language feature**
  - Not based on (simple) method patterns…
  - … but can be intercepted via the DLR
- **IDynamicMetaObjectProvider**
  - Object used by the DLR for late binding
  - Provides access to a DynamicMetaObject
    - Very powerful (see Part 2 of this course)
    - Quite complex to implement
- **DynamicObject**
  - Reduces the amount of plumbing needed

Dynamic Typing



Awaitable Types

- **Await expressions**
  - Suspend the asynchronous method they're contained in…
  - … until the awaitable object signals completion, eventually

```
async Task<double> SurfaceAsync(double r)
{
  var pi = await ComputePiAsync();
  return pi * r * r;
}
```

Operand of await should be awaitable

Covered in Part 2 of this course

- **Mechanism**
  - Asynchronous method chopped up in a state machine
    - Each await expression introduces a state transition
    - Return and throw wired up to resulting task
  - Await expressions
    - Method pattern for obtaining an **awaiter**
    - Continuation registered to trigger "move next" on state machine

Awaitable Types

- Compilation of the async method (simplified)

```
async Task<double> SurfaceAsync(double r) {
  var tcs = new TaskCompletionSource<double>();
  var state = 0;
  var awaiter1 = default(TaskAwaiter<double>);
  var moveNext = new Action(() => {
    switch (state) {
      case 0: awaiter1 = ComputePiAsync().GetAwaiter();
              state = 1;
              if (awaiter1.IsCompleted) goto case 1;
              else awaiter1.OnCompleted(moveNext);
              break;
      case 1: tcs.SetResult(awaiter1.GetResult());
              break;
    }
  });
  moveNext();
  return tcs.Task;
}
```

Really using an AsyncMethodBuilder

Really using an IAsyncStateMachine

Omitted some error handling

Awaitable Types

- **An object of type T is awaitable if**
  - The compiler can emit a **GetAwaiter()** method call, with result type A
    - Instance method, extension method, or dynamically typed
  - If the awaiter type A is dynamic, or
    - Has a Boolean-returning **IsCompleted** property
    - Has a **GetResult()** method returning void or some type R
    - Implements **INotifyCompletion** (or ICriticalNotifyCompletion)

```
namespace System.Runtime.CompilerServices
{
  public interface INotifyCompletion
  {
    void OnCompleted(Action continuation);
  }
}
```

Delegate to the state machine

- **Task<T> and Task implement the pattern**
  - TaskAwaiter<T> and TaskAwaiter structs

Awaitable Types

- Awaiting a button click

```
static class ButtonExtensions {
  public static ButtonClickAwaiter GetAwaiter(this Button b) {
    return new ButtonClickAwaiter { Button = b };
  }
}
```

```
class ButtonClickAwaiter : INotifyCompletion {
  public Button Button;

  public bool IsCompleted { get { return false; } }

  public void OnCompleted(Action continuation) {
    EventHandler h = null;
    h = (o, e) => {                          continuation(); };
    Button.OnClick += h;
  }
}
```

No synchronous completion

Don't leak!

Summary

- **Dynamic typing**
  - "dynamic" in C# 4.0 is a static type
  - Interactions with the DLR
    - Language binders
    - Call site caching
  - Using DynamicObject to intercept calls

- **Awaitable types**
  - Operand of await expressions
  - Pattern based
    - GetAwaiter, IsCompleted, GetResult, OnCompleted
  - State machine primer