# Module 12: "Flyweight"

# Agenda

- Introductory Example: Brewing Coffee
- Challenges
- Implementing the Flyweight Pattern
- Pattern: Flyweight
- Overview of Flyweight Pattern
- .NET Framework Example: String Interning
- Additional Comments on Flyweight

# Introductory Example: Brewing Coffee

```
interface ICoffee
{
    CoffeeKind Kind { get; }
    int Strength { get; }
    CoffeeSize Size { get; }
    string CustomerName { get; }

    void Serve();
}
```

```
class Cappuccino : Coffee
{
    public Cappuccino( string name )
        : base(
          CoffeeKind.Cappuccino,
          3,
          CoffeeSize.Regular,
          customerName) { }
}
```

```
ICoffee c1 = new Cappuccino("John Doe");
c1.Serve();
ICoffee c2 = new Espresso("Jane Doe");
c2.Serve();
```

# Challenges

- Very many (nearly) identical objects are created
- But how could that be avoided when customer names varies..?

- Can we separate objects' state into
  - Shared +
  - Non-shared?

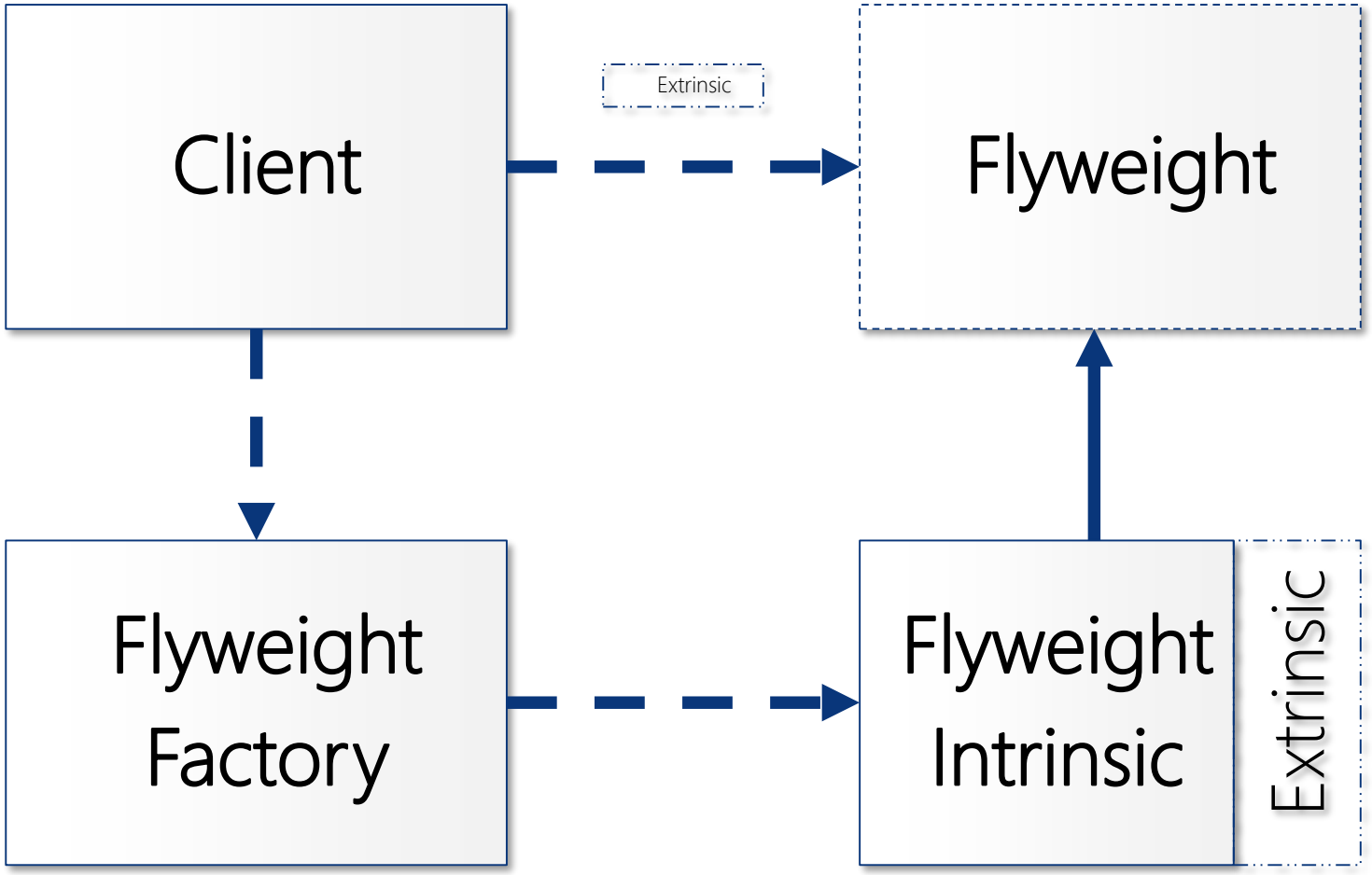
# Pattern: Flyweight

- *Use sharing to support large numbers of fine-grained objects efficiently.*

- Outline
  - Creating a large number of objects should be avoided
  - Store *"intrinsic"* (i.e. invariant) state that can be shared
  - Allow *"extrinsic"* (i.e. variant) state to be passed in methods

- Origin:
  - Paul Calder and Mark Linton (1990) (also in Gang of Four)

# Overview of Flyweight Pattern



# Overview of Flyweight Pattern

- Client
  - Gets Flyweight Intrinsic object from Flyweight Factory
  - Invokes operations on Flyweight Intrinsic providing extrinsic state
- Flyweight Factory
  - Cache for Flyweight objects
- Flyweight
  - Interface or abstract base class for concrete Flyweight objects
- Flyweight Intrinsic
  - Concrete Flyweight class storing intrinsic state
  - Immutable value object

# .NET Framework Example: String Interning

- Strings in .NET are interned as flyweights
  - Intrinsic state only

```
string s1 = "Hello";
string s2 = "Hello";
string s3 = Console.ReadLine();
string s4 = string.Intern( s3 );

Console.WriteLine( ReferenceEquals( s1, s2 ) );
Console.WriteLine( ReferenceEquals( s1, s3 ) );
Console.WriteLine( ReferenceEquals( s1, s4 ) );
```

# Additional Comments on Flyweight

- Mostly applicable to very specific scenarios
  - Frameworks
  - Graphics and rendering
- Not used very frequently nowadays in business code
- Flyweights should be immutable for safe sharing
- Should flyweight be structs?
- What about thread-safety?


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Hasselvangen 243
8355 Solbjerg
Denmark