

Module 7

"Properties and Static Methods"



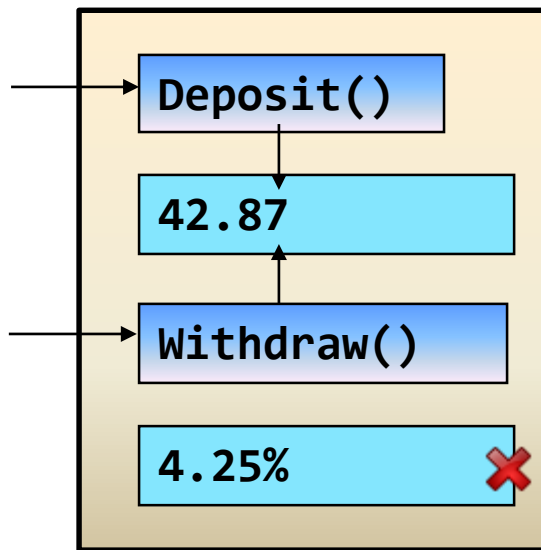
TEKNOLOGISK
INSTITUT

Agenda

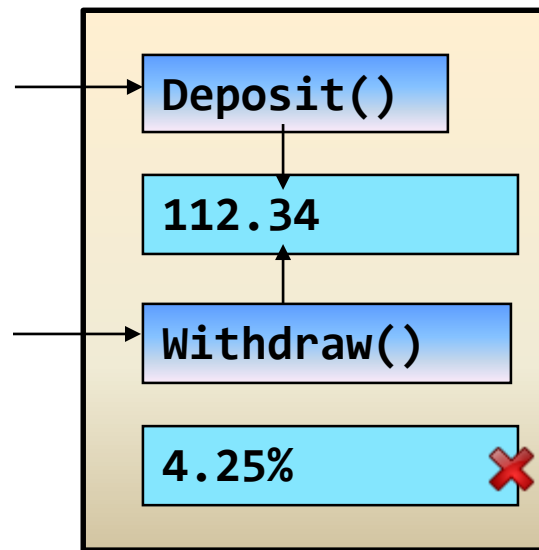
- ▶ Static Classes and Members
- ▶ Properties and Initializers
- ▶ Lab 7
- ▶ Discussion and Review

Introducing Static Data

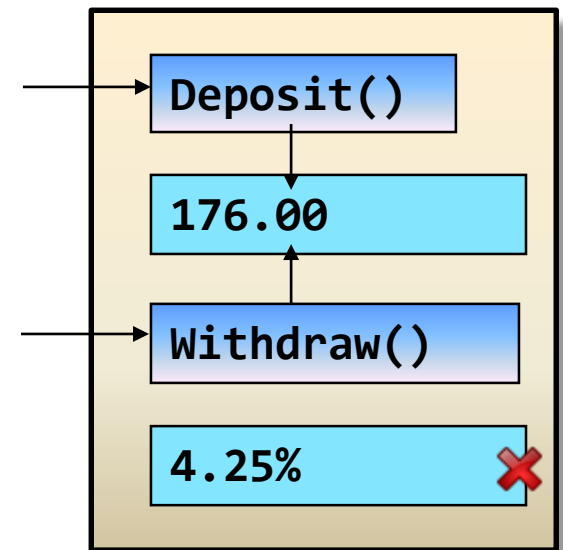
- ▶ Static data captures information shared between all the objects of a class



BankAccount



BankAccount



BankAccount

Static Data

- ▶ With instance data each object maintains an independent copy
- ▶ Class data can be *static*, i.e. shared among all instances

```
class BankAccount
{
    private decimal _currentBalance;
    public static decimal CurrentInterestRate = 0.04m;

    public BankAccount( decimal balance )
    {
        _currentBalance = balance;
    }
}
```

- ▶ Refers to the same physical in-memory location!



Static Methods

- ▶ Static data should be manipulated by static methods

```
class BankAccount
{
    ...
    public static decimal CurrentInterestRate = 0.04m;

    public static void SetInterestRate( decimal interestRate )
    {
        CurrentInterestRate = interestRate;
    }
}
```

- ▶ Invoke static methods via class name instead of instance name!

```
BankAccount.SetInterestRate( 0.06m );
```



Static Constructors

- ▶ Initializing static data should be done in static constructors

```
class BankAccount
{
    public static decimal CurrentInterestRate;

    static BankAccount()
    {
        CurrentInterestRate = 0.04m; // This could be dynamic!
    }
}
```

- ▶ Only one static constructor for each class
- ▶ Has no access modifier and no parameters
- ▶ Invoked by the runtime system before first instance constructor
- ▶ Invoked exactly once regardless of number of objects created



Static Classes

- ▶ Classes themselves can also be static

```
static class TimeUtility
{
    public static void PrintTime()
    {
        Console.WriteLine( DateTime.Now.ToShortTimeString() );
    }
    public static void PrintDate()
    {
        Console.WriteLine( DateTime.Today.ToShortDateString() );
    }
}
```

- ▶ Cannot be instantiated
- ▶ Can only contain static fields and methods

```
TimeUtility tu = new TimeUtility();
```



Static Usings

- ▶ Static members can now be imported with **using static**

```
using static System.Console;  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        WriteLine( "Hello, World!" );  
    }  
}
```



Agenda

- ▶ Static Classes and Members
- ▶ **Properties and Initializers**
- ▶ Lab 7
- ▶ Discussion and Review

Properties

- ▶ Encapsulation is achieved by *Properties*

```
class Button
{
    public string Caption
    {
        get { return _caption; }
        set { _caption = value; }
    }
    private string _caption;
}
```

```
Button button = new Button();
button.Caption = "Click!!";
```

```
Console.WriteLine( button.Caption );
```

- ▶ Two specific accessors
 - get is invoked when retrieving the value
 - set is invoked when setting the value



Visibility of Get/Set

- ▶ Access modifiers can be set for **get** and **set** separately

```
class Button
{
    public string Caption
    {
        get { return _caption; }
        private set { _caption = value; }
    }
    private string _caption;
}
```

```
Button button = new Button();
Console.WriteLine( button.Caption ); ✓
button.Caption = "Click!!"; ✗
```

Read-Only and Write-Only Properties

- ▶ Property can be made read-only by omitting **set**
- ▶ Property can be made write-only by omitting **get**

```
class Button
{
    public string Caption
    {
        get { return _caption; }
        // No set!
    }
    private string _caption;
}
```

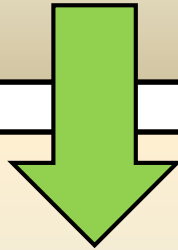
```
class Login
{
    public string Password
    {
        // No get!
        set { _password = value; }
    }
    private string _password;
}
```

- ▶ Properties can also be static

Defining Automatic Properties

- Automatic properties ease the burden of defining “trivial” properties

```
class Car
{
    public string PetName
    {
        get { return _petName; }
        set { _petName = value; }
    }
    private string _petName = string.Empty;
}
```



```
class Car
{
    public string PetName { get; set; }
}
```



Auto-property Initializers

- ▶ Since C# 6.0 Initializers can be supplied for the automatic properties

```
class Car
{
    public string PetName { get; set; } = "Chuck";
    public int CurrentSpeed { get; set; } = 0;
}
```

- ▶ Otherwise, default value of an automatic property is the usual "zero-whitewash"
 - Reference types are null
 - Integers are 0
 - Booleans are false
 - ...
- ▶ Note: Structs still cannot have initializers!



Getter-only Auto-properties

- ▶ Moreover, the automatic properties can in C# 6.0 be getter- or setter-only

```
class Car
{
    public string PetName { get; } = "Chuck";
    public int CurrentSpeed { get; set; } = 0;
    public DateTime LastUpdated { get; } = DateTime.Now;
}
```

- ▶ This is a really important mechanism for putting mutable and immutable data types on equal terms!
- ▶ Underlying field is created as **readonly**
 - Can still be assigned from the constructor
 - ... but elsewhere not!
- ▶ Note: Can still have initializers!



Restricting Access to Automatic Properties

- ▶ Use access modifiers on at most one(!) of the get or set accessors

```
class Car
{
    public string PetName { private get; private set; } ❌
}
```

```
class Car
{
    public string PetName { get; private set; } ✅
}
```

- ▶ Note: Neither get nor set can be more visible than the parent property



Object Initializer Syntax

- ▶ Object initializer syntax can be used to assign values for public properties and fields during construction

```
Point p = new Point { X = 42, Y = 87 };  
Console.WriteLine( "p is {0}", p );
```

- ▶ Custom constructors can be invoked as well

```
Point q = new Point( 16, 24 ) { X = 112 };  
Console.WriteLine( "q is {0}", q );
```

- ▶ Object initializers execute after constructors
- ▶ Object initializers can initialize any subset of available properties and fields



Initializing Inner Types and Collections

- ▶ Inner types can now be conveniently initialized

```
public class Rectangle
{
    public Point TopLeft { get; set; }
    public Point BottomRight { get; set; }
    ...
}
```

```
Rectangle r = new Rectangle
{
    TopLeft = new Point { X = 10, Y = 10 },
    BottomRight = new Point { X = 90, Y = 90 }
};
Console.WriteLine( r );
```



Constant Data

- ▶ Data is deemed constant by using the **const** keyword

```
class MyMathClass  
{  
    public const double Pi = 3.14;  
}
```

```
Console.WriteLine( MyMathClass.Pi ); ✓
```

```
MyMathClass.Pi = 22 / 7; ✗
```

- ▶ Such data cannot be changed!
- ▶ Curious fact: Constant fields are implicitly static



Read-only Data

- Read-only data can only be set in constructors

```
class MyMathClass
{
    public MyMathClass()
    {
        if( DateTime.Today.Day % 2 == 0 )
        {
            TodaysPi = 3.14;
        }
        else
        {
            TodaysPi = 22.0 / 7;
        }
    }
    public readonly double TodaysPi;
}
```

```
MyMathClass m = new MyMathClass();
Console.WriteLine( m.TodaysPi );
```



```
m.TodaysPi = 4.00;
```



```
public void SetTodaysPi( double tp )
{
    TodaysPi = tp;
}
```



Methods vs. Properties

- ▶ Properties are somewhere in between public member variables and methods
- ▶ Methods
 - Defined and invoked using parenthesis
 - Might take parameters
- ▶ Properties
 - Defined and invoked without parenthesis
 - No additional parameters: Gets or sets a single value

```
class BankAccount
{
    ...
    public decimal GetBalance()
    {
        return _balance;
    }
}
```

```
BankAccount ba = ...;
decimal d = ba.GetBalance();
```


```
class BankAccount
{
    ...
    public decimal Balance
    {
        get { return _balance; }
    }
}
```

```
BankAccount ba = ...;
decimal d = ba.Balance;
```



Quiz: Properties and Static Members – Right or Wrong?


```
class Car
{
    public static int SpeedLimit;
    public string PetName;
    public int CurrentSpeed;
}
```




```
Car c;
c.SpeedLimit = 50;
```




```
Car c = new Car();
c.SpeedLimit = 50;
```




```
Car.SpeedLimit = 50;
```



```
class Point
{
    public int X { get; set; }
    public int Y { get; set; }
}
```




```
Point p = new Point();
p.X = 42;
p.Y = 87;
```



```
Point p = new Point{ X = 42 };
```



```
class Person
{
    public int Id { private get; }
}
```





Lab 7: Encapsulating Data



Discussion and Review

- ▶ Static Classes and Members
- ▶ Properties and Initializers



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Hasselvangel 243

8355 Solbjerg

Denmark