TEKNOLOGISK
INSTITUT

# Module 7

# Properties and Static Methods

# Properties

- Encapsulation is achieved by *Properties*
  - Compiled down to methods
- Two specific accessors
  - get is invoked when retrieving the value
  - set is invoked when setting the value
  - Can be used individually
    - Read / Write only
- Why not
  - use public fields
  - use read only public fields
  - use methods to control access
- Because
  - Better syntax (declaration and use of properties)
  - Separation of data and action
    - Methods = action, Properties = data
  - Databinding (INotifyPropertyChanged)

TEKNOLOGISK
INSTITUT

# Syntax

- Simple syntax
  - Private (backing) field
  - Property with get/set

```
class Person {

    private string name;

    public string Name
    {
        get {
            return name;
        }
        set {
            name = value;
        }
    }
}
```

TEKNOLOGISK
INSTITUT

# Read-Only and Write-Only Properties

- Property can be made read-only by omitting set
- Property can be made write-only by omitting get
- Use the private keyword for internal access

```
private string name;

public string Name
{
    get {
        return name;
    }
}
```

```
private string name;

public string Name
{
    get {
        return name;
    }
    private set {
        name = value;
    }
}
```

```
private string name;

public string Name
{
    set {
        name = value;
    }
}
```

TEKNOLOGISK
INSTITUT

# Defining Automatic Properties

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

```
class Person {

    private string name;

    public string Name
    {
        get {
            return name;
        }
        set {
            name = value;
        }
    }
}
```

```
class Person {
    public string Name { get; set; }
}
```

TEKNOLOGISK
INSTITUT

# Auto-property Initializers

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

```
class Person {
    public string Name { get; set; } = "";
    public int Age { get; } = 0;    // default
    public Gender Gender { get; private set; } = Gender.Male;
}
```

TEKNOLOGISK
INSTITUT

# Getter-only Auto-properties

- Moreover, the automatic properties can in C# 6.0 be getter- or setter-only and use the private keyword

```
class Person {
    public string Name { get; set; }
    public int Age { get; }
    public Gender Gender { get; private set; }
}
```

TEKNOLOGISK
INSTITUT

# Object Initializer Syntax

- Object initializer syntax can be used to assign values for public properties and fields during construction
- Custom constructors can be invoked as well
- Object initializers execute after constructors
- Object initializers can initialize any subset of available properties and fields

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public Gender Gender { get; set; }

    public Person()
    {
        this.Name = "";
        Gender = Gender.Male;
    }
}
```

```
Person p = new Person()
{
    Name = "Mikkel",
    Age = 14,
    Gender = Gender.Male
};
```

TEKNOLOGISK
INSTITUT

# 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!

TEKNOLOGISK
INSTITUT

# Static Methods

- Static data should be manipulated by static methods

```
class BankAccount
{
    public int Id { get; set; }
    public static decimal CurrentInterestRate = 0.04m;
    public static void SetInterestRate(decimal interestRate)
    {
        CurrentInterestRate = interestRate;
    }
    public static BankAccount GetNewAccount() {
        // get from db...
        return new BankAccount() { Id = 1 };
    }
}
```

```
BankAccount.SetInterestRate(.01m);
```

TEKNOLOGISK INSTITUT

# 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

TEKNOLOGISK
INSTITUT

# 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

TEKNOLOGISK
INSTITUT

# 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!");
    }
}
```