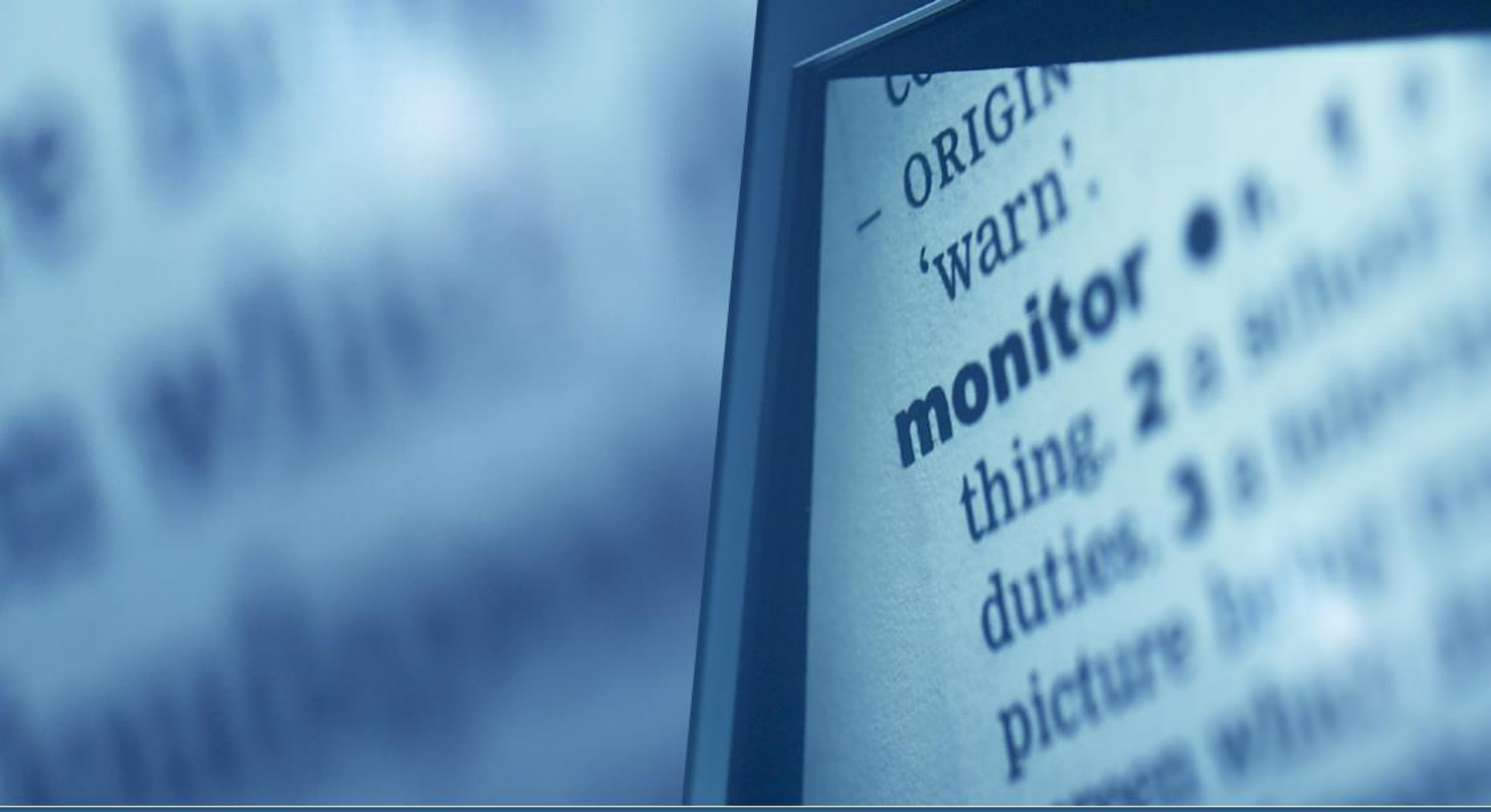

Programming 3

Scope: access modifiers

- Each member of field and method in a class has a specific 'accessibility'
- This accessibility can be set with so-called 'access modifiers'
  - public → field/method is available in its own class, in derived classes and outside the class;
  - private → field/method is only available in its own class;
  - protected → field/method is only available in its own class and in derived classes;

Public fields/methods

- Public fields/methods can be used everywhere

```
class Book {
    // member fields
    public string title;
    public float price;
    public int amount;

    // constructor
    1 reference
    public Book(string title, float price, int amount) {
        this.title = title;
        this.price = price;
        this.amount = amount;
    }
}
```

```
static void Main(string[] args) {
    Book book1 = new Book("Joe speedboat", 10.00f, 5);
    book1.amount = 6;
    book1.price = 8.99f;
}
```

Private fields/methods

- Private fields/method can only be used in the class itself (encapsulation)

```
class Book {
    // member fields
    public string title;
    public float price;
    private int amount;

    // constructor
    1 reference
    public Book(string ti
        this.title = titl
        this.price = pric
        this.amount = amo
    }
}
```

```
public void ChangeStock(int amount) {
    if (amount >= 0) {
        this.amount = amount;
    }
}

0 references
public int GetAmount() {
    return amount;
```

```
class Program {
    0 references
    static void Main(string[] args) {
        Book book1 = new Book("Joe speedboat", 10.00f, 5);
        book1.amount = 6;
        book1.price = 8.99f;
    }
}
```

Private fields/methods

```
class Magazine : Book {
    // member fields
    public DayOfWeek publishingDay;

    // constructor
    0 references
    public Magazine(string title, float price, int amount, DayOfWeek publishingDay)
        : base(title, price, amount) {
        this.publishingDay = publishingDay;
    }

    0 references
    public override string ToString() {
        return "[Magazine] \"" + title + "\", " + price.ToString("0.00") + ",  " + amount;
    }
}
```

A derived class also has no access to private members in the base class.

Protected fields/methods

- Protected fields/methods are in the class itself and can be used in derived classes

```
class Book {
    // member fields
    public string title;
    public float price;
    protected int amount;

    // constructor
    2 references
    public Book(string title, float price, int amount) {
        this.title = title;
        this.price = price;
        this.amount = amount;
    }

    0 references
    public void ChangeStock(int amount) {
        if (amount >= 0) {
            this.amount = amount;
        }
    }

    0 references
    public int GetAmount() {
        return amount;
    }
}
```

```
class Magazine : Book {
    // member fields
    public DayOfWeek publishingDay;

    // constructor
    0 references
    public Magazine(string title, float price, int amount, DayOfWe

    0 references
    public override string ToString() {
        return "[Magazine] \"" + title + "\", "
            + price.ToString("0.00") + ",  "
            + amount.ToString();
    }
}
```

Properties

- Properties are fields with ‘access facility’
- The property consists of a set (assign) and a get (read) method/assessor
- Properties without a set accessor are read-only
- Properties without a get accessor are write-only
- Properties ‘should be lightweight’

Properties – example 1

```
class Book {
    public string title;

    0 references
    public string Title {
        get { return title; }
        set { title = value; }
    }
}
```

Additional advantage: you can now also place a breakpoint, for example, if you want to know when a field is modified.

```
static void Main(string[] args) {
    Book book1 = new Book();
    // when writing the Title property, the set accessor is called
    book1.Title = "Joe speedboat II";

    // when reading the Title property, the get accessor is called
    Console.WriteLine("Title of the book is {0}", book1.Title);
}
```

Properties – example 2

```
class Book {
    private string title;
    private int amount;

    0 references
    public int Amount {
        get { return amount; }
        set { amount = value; }
    }

    2 references
    public string Title {
        get { return title; }
        set { title = value; }
    }
}
```

```
class Book {
    private string title;
    private int amount;

    0 references
    public int Amount {
        get { return amount; }
        set {
            if (value >= 0) {
                amount = value;
            }
        }
    }

    2 references
    public string Title {
        get { return title; }
        set { title = value; }
    }
}
```

Auto-implemented properties

- Auto-implemented properties: abbreviated notation, no explicit member fields

```
class Book {
    // automatic properties
    3 references
    public string Title { get; set; }
    1 reference
    public float Price { get; set; }
    1 reference
    public float Amount { get; set; }

    // constructor
    2 references
    public Book(string title, float price, int amount) {
        this.Title = title;
        this.Price = price;
        this.Amount = amount;
    }
}
```

Internal modifications (in the future) will not change the interface!!

So, if e.g. property Price gets extra code in the set, it will not influence the 'outside world'.

Read-only properties

```
class Book {
    // automatic properties
    3 references
    public string Title { get; private set; }
    1 reference
    public float Price { get; set; }
    1 reference
    public float Amount { get; set; }

    // constructor
    2 references
    public Book(string title, float price, int amount) {
        this.Title = title;
        this.Price = price;
        this.Amount = amount;
    }
}
```

Property Title is read-only (for the outside world), since the set can only be used inside the class.

Title can be changed in e.g. the constructor (since it's <u>inside</u> the class).

Calculated properties

```
class Book {
    // automatic property
    2 references
    public string Title { get;  set; }

    // backing field for property Amount
    private int amount;

    // property
    1 reference
    public int Amount {
        get {
            return amount;
        }

        set {
            if (value >= 0) {
                amount = value;
            }
        }
    }

    // read only property
    1 reference
    public float Price { get; private set; }

    // Computed property
    0 references
    public float TotalValue {
        get {
            return Price * Amount;
        }
    }
}
```

calculated property: other properties are used here (to determine the return value).

Homework (for next week)

- Read paragraphs ‘Yellow Book’
  (references can be found on Moodle)

- Assignments week 3
  (can be found on Moodle)