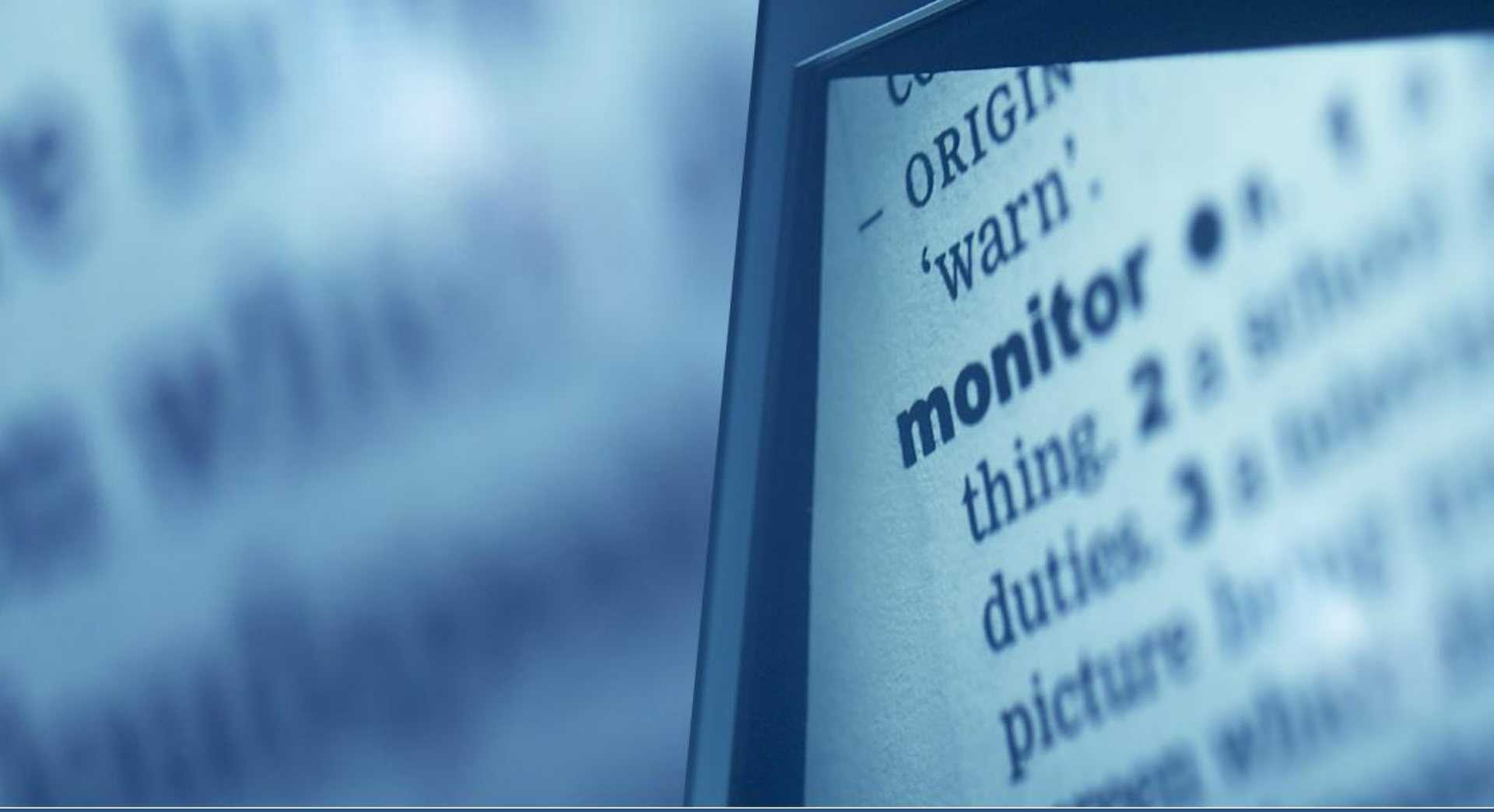

Programming 2

Program term 1.2

01 (wk-46)	enumerations / structures / classes
02 (wk-47)	2-dim arrays / flow control
03 (wk-48)	lists / dictionaries
04 (wk-49)	file I/O / error handling
05 (wk-50)	Class Libraries / Layered Architecture
06 (wk-51)	SoC / SRP
07 (wk-52)	Christmas holiday
08 (wk-53)	Christmas holiday

---

09 (wk-01)	practice exam
10 (wk-02)	*exams*
11 (wk-03)	*retake exams*
12 (wk-04)	*retake exams*

Any questions about weekly assignments?

Separation of Concerns (SoC)

Separation of Concerns (SoC)

- In computer science, Separation of Concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern.

- http://en.wikipedia.org/wiki/Separation_of_concerns

Separation of Concerns - methods

- There are different levels of SoC
- We can create **separate methods** for dealing with separate concerns/functionalities
- In the P2 assignments we have used separate methods for reading, displaying, writing, calculating, …

  - reading a Person from the user → `Person ReadPerson()`
  - displaying a Person to screen → `DisplayPerson(Person p)`
  - reading a Person from file → `Person ReadPerson(string filename)`
  - writing a Person to file → `WritePerson(Person p, string filename)`
  - …

Separation of Concerns - classes

- We can also create **separate classes** for dealing with separate concerns/functionalities

- We have used a separate class for the state of the Hangman game (`class HangmanGame, week 3`)

  - storing secret word and guessed word
  - initializing these 2 words → `Init(string secretWord)`
  - processing a letter → `void ProcessLetter(char letter)`
  - determining if the word has been guessed? → `bool IsGuessed()`

Separation of Concerns - layers

- We can also create **separate layers** for dealing with separate concerns/functionalities

- In the next term we will use 3 separate layers:

1. presentation layer: concerned with communication with the user (reading / displaying)
2. logic layer: concerned with the (business) logic of the application *(like the game logic of CandyCrush: methods* `RowScorePresent` *and* `ColumnScorePresent`*)*
3. data access layer: concerned with how data is stored and retrieved

Lingo game

- Last week we showed pseudo code of the Lingo game
- What did we do, in order to meet the SoC principle?
- What different 'concerns' (functionalities) does the Lingo game have?
    1. *user communication*
    2. *game state/logic, …*

Lingo game

- We added a class LingoGame that stores the lingo word and the player word, and determines the state of each letter (in the player word)
- Class Program is concerned with user communication (reading next player word, displaying results, …)
- Class LingoGame is concerned with the state of the game (lingo word / player word), and the state of the letters in the player word

class LingoGame

Class LingoGame is concerned with the state of the Lingo game

Storing the lingo word and playerword

Number of attempts left is also part of the state…

```
class LingoGame
        lingoWord
        playerWord
        attemptsLeft ← we could add this...

        Init(theLingoWord)
                lingoWord = theLingoWord
                attemptsLeft = 5     ← and this...

        GuessedWord()
                return lingoWord = playerWord

        HasAttemptsLeft()            ← and this method...
                return attemptsLeft > 0
```

class LingoGame

Class LingoGame is concerned with the state of the Lingo game

```
class LingoGame
        // continued...

        ProcessWord(thePlayerWord)
                playerWord = thePlayerWord
                attemptsLeft = attemptsLeft - 1 ← and this...

                // code to determine state of the letters...
                ...


                return letterResults
```

The lingo game can determine the letter results of the player word

class Program

Class Program is concerned with user communication

```
class Program

Start()
        words = ReadWords()
        lingoWord = SelectWord(words)

        LingoGame game = new LingoGame()
        game.Init(lingoWord)
        PlayLingo(game)

ReadWords()
        return new string[] { "green", "class", "school" };
        // or read words from a file...

SelectWord(words)
        return words[random.Next(words.Count)]
```

class Program

Class Program is concerned with user communication

```
PlayLingo(game)
        while not game.GuessedWord() and
                                        game.HasAttemptsLeft()
                playerWord = ReadPlayerWord(5)
                results = game.ProcessWord(playerWord)
                DisplayResults(results)
```

Processing of the (user) word is done by the Lingo game...

```
ReadPlayerWord(length)
        // read player word...
```

User communication: read next player word, display results, ...

```
DisplayResults(playerWord, results[])
        // display letter results...
```

Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP)

- The Single Responsibility Principle (SRP) is a computer programming principle that states that every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.
- This also applies to methods!
- https://en.wikipedia.org/wiki/Single_responsibility_principle

Single Responsibility Principle (SRP)

- An as example we will take a look at **on old method** of class Hangman:

```
bool CheckLetter(char letter)
```

- *This method checks if the secret word contains the given letter, and returns true if it does, false otherwise. If the secret word contains the letter, then change the guessed word by putting this letter in the correct place(s).*

bool CheckLetter(char letter)

```
bool CheckLetter(letter)
        if not secretWord.Contains(letter)
                return false

        string newGuessedWord = ""
        for (i = 0 to secretWord.Length - 1)
                if (secretWord[i] = letter)
                        newGuessedWord = newGuessedWord + letter
                else
                        newGuessedWord = newGuessedWord +
                                                guessedWord[i]
        guessedWord = newGuessedWord
        return true
```

`bool CheckLetter(char letter)`

- What's is wrong with this method (see pseudo code on the previous slide), if we think about the Single Responsibility Principle (SRP)?

- It does 2 things:
  1) determine if the (secret) hangman word contains the given letter (and returns true or false)
  2) update the guessed word

- It is better to split up this method into 2 separate methods:
  1) `bool ContainsLetter(letter)`
  2) `void ProcessLetter(letter)`

bool ContainsLetter(char letter)

```
bool ContainsLetter(letter)
        foreach (swLetter in secretWord)
                if swLetter = letter
                        return true
        return false
```

or…

```
bool ContainsLetter(letter)
        return secretWord.Contains(letter)
```

void ProcessLetter(char letter)

```
void ProcessLetter(letter)

        string newGuessedWord = ""
        for (i = 0 to secretWord.Length - 1)
                if (secretWord[i] = letter)
                        newGuessedWord = newGuessedWord + letter
                else
                        newGuessedWord = newGuessedWord +
                                                guessedWord[i]

        // update guessed word
        guessedWord = newGuessedWord
```

Hangman loop

```
List<char> enteredLetters = new List<char>()
int attemptsLeft = 8
while attemptsLeft > 0 and not hangman.SecretWordGuessed()
        letter = ReadLetter(enteredLetters)
        enteredLetters.Add(letter)
        DisplayLetters(enteredLetters)

        if hangman.ContainsLetter(letter)
                hangman.ProcessLetter(letter)
        else
                attemptsLeft = attemptsLeft - 1

        Display attemptsLeft
        DisplayWord(hangman.guessedWord)
```

Are there other things we can improve…?

Homework

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

- Assignments week 6
  (can be found on Moodle)