SIT771 – Lecture 1

Introduction to object-oriented programming

DEAKIN UNIVERSITY

Further reading

- Paul Deitel and Harvey Deitel (2018). Visual C# how to Program (6th ed). Pearson. Ebook on Deakin Library – Chapter 3 and Chapter 4.

Outline

In this lecture...

- Unit organization
- Assessment
- Study structure
- Introduction to programming
- Introduction to object-oriented programming (OOP)
  - Main OOP concepts
  - Sample scenario
  - Ingredients of OOP

UNIT ORGANIZATION

Teaching team

Tutors/OnTrack Markers

- Geelong Waurn Ponds + OnTrack Marker; Ms. Huyen Tran (Tuesday 15.00-16.50)
- Geelong Waurn Ponds + OnTrack Marker; Ms. Huyen Tran (Thursday 15.00-16.50)
- Burwood + OnTrack Marker; Mr. Durgesh Samariya (Wednesday 8.00-9.50)
- Burwood + OnTrack Marker; Ms. Bansri Modi (Wednesday 15.00-16.50)
- Burwood + OnTrack Marker; Ms. Bansri Modi (Thursday 12.00-13.50)
- IT HelpHub; Ms. Olivia McKeon (limited sessions)

Unit Chair, Lecturer, and Online Workshop Tutor

- Dr. Bahadorreza Ofoghi; Senior Lecturer, Information Technology
- Background in natural language processing, information retrieval, and machine learning
- Email: b.ofoghi@deakin.edu.au

Interaction

Lectures

- For all students: 1 x 1 hour class per week [online – MS Teams]

Workshops

- For on-campus students: 1 x 2 hour per week
- For online students: 1 x 2 hour per week – Wednesday 18.00-20.00 AEST [online – MS Teams]

Burwood

SIT771_B_D_T2 - OBJECT-ORIENTED DEVELOPMENT

Comments:

Type/Stream/Recorded		Day	Start/End	Campus	Location	Teaching weeks
Lecture 1 - Online	01	Mon	09:00-09:50			01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11
Workshop 1 - BYOD	01	Wed	08:00-09:50	Burwood - Elgar Road	LC6.109	01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11
Workshop 1 - BYOD	02	Wed	15:00-16:50	Burwood - Elgar Road	LC3.101	01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11
Workshop 1 - BYOD	03	Thu	12:00-13:50	Burwood	B1.28	01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11

Geelong, Waurn Ponds

SIT771_G_D_T2 - OBJECT-ORIENTED DEVELOPMENT

Comments:

Type/Stream/Recorded		Day	Start/End	Campus	Location	Teaching weeks
Lecture 1 - Online	01	Mon	09:00-09:50			01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11
Workshop 1 - BYOD	02	Tue	15:00-16:50	Waurn Ponds	KA4.413	01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11
Workshop 1 - BYOD	01	Thu	15:00-16:50	Waurn Ponds	KA4.405	01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11

Materials

Available on the following platforms

- CloudDeakin
  - Unit information
  - Links to readings
  - Lecture slides, demos, weblinks
  - Discussion forums
- OnTrack
  - Task definitions, resources, and learning outcomes
  - Assignment submission and feedback
  - Monitor progress
- MS Teams
  - Lecture/workshop recordings

Weekly subjects

Week #	Topic
1	Introduction to object-oriented programming
2	Classes, objects
3	Control flow, error detection, error handling
4	Modelling, object/class relations
5	Useful data structures and code evaluation
6	Responsibilities and design considerations
7	Polymorphism
8	Interface, delegation
9	Another programming language
10	Revision and the way forward
11	–

Image source:
https://www.pnas.org/content/118/3/e2024425118

ASSESSMENT

Portfolio-based evaluation

Portfolio – 100%

- Find tasks, note the given grades per task, and submit your work on OnTrack
- Take note of submission deadlines
  - Apply for an extensions **before** the task submission deadline
  - **Time exceeded** tasks will get **no feedback** or approval until the end of trimester
  - **Time exceeded** tasks may receive a **Fail** outcome (if erroneous) as you will not get a chance to fix them

- To achieve a specific target grade (e.g., D), you must:
  - Complete and submit all tasks with lower grades (P and C in this case) **on-time**, and achieve the grades
  - Complete and submit all tasks of your specific target grade (D in this case) **on-time**, and achieve the grade
  - The above completions **will not necessarily** mean that you will achieve the **target grade**. Other criteria include:
    - quality and timeliness of submissions
    - marks in graded tasks

Portfolio-based evaluation

Portfolio – 100% (cont.)

- Familiarize yourself with OnTrack outcomes
- Expect feedback within **3-4 business** days
- Move with the pace of the trimester/class
- **DO NOT FORGET** to submit your **portfolio**; a separate completion

- **Fail**: Not successfully met the requirements of the task.
- **Feedback Exceeded**: Task has been submitted too many times, with the teaching staff concluding that the student is not heeding feedback.
- **Redo**: The task has been assessed to be of low quality, or inappropriate, and should be restarted.
- **Fix and Resubmit**: The task is on the right track, but needs to be improved or fixed.
- **Discuss**: The tutor is happy with the submission, and would like to discuss the task with the student.
- **Demonstrate**: The tutor is happy with the submission, and would like the student to demonstrate the work.
- **Complete**:The tutor is happy with the submission, and it is ready for inclusion in the portfolio.

Image source: https://doubtfire.io/guides/glossary

Plagiarism

What is considered plagiarism?

- Plagiarism includes the following:
  - Copying another person's ideas or expressions without appropriate acknowledgment and presenting their ideas or forms of expressions as your own.
  - Written works such as books or journals as well as data or images that may be presented in tables, diagrams, designs, plans, photographs, films, music, formulae, web sites, and computer programs.
  - The use of (or passing off) the work of lecturers or other students as your own.

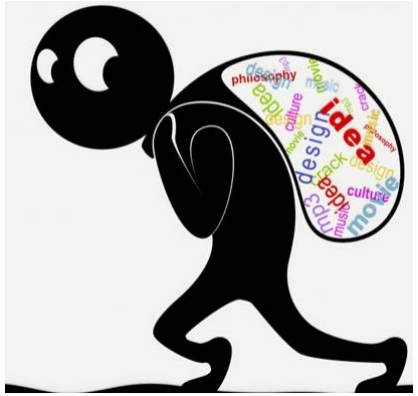


Image source: New Age | Plagiarism in education and society (newagebd.net)

Plagiarism

Negative outcomes

- If the Faculty Academic Progress and Discipline Committee (FAPC) finds a student has committed an act of academic misconduct, it may impose one or more of the following penalties:
  - Allocate a zero mark or result or other appropriate mark or result for the assessment task;
  - Allocate a zero mark or result or other appropriate mark or result for the Unit;
  - Suspend from a Unit or a Course for up to 4 Study Periods;
  - Exclude from the University;
  - Pay the cost of investigating the misconduct;
  - Require the Student to refrain from association with specified person/s for purposes of study or assessment;
  - Reprimand and caution the student;
  - Require resubmission of one or more assessment tasks;
  - Require a student to undertake alternative assessment for the Unit on terms determined by the faculty committee;
  - Terminate candidature; Recommend to the vice-chancellor or nominee that the degree not be awarded

STUDY STRUCTURE

Collaborative learning

Working with other students in this unit

- You may discuss/collaborate with other students to better understand a problem and to determine an approach to solving the problem.
- You are not permitted to share your solutions (whether finished or in progress) with other students under any circumstances.
- Your assignment submission, i.e., code, documentation, etc., must be entirely your own work.

Referencing sources

- Any code that has been copied/adapted should be clearly referenced (including any code from assignment questions).
- **Note:** there should be little need for the above; otherwise, you are deemed as not learning the content well enough to pass the unit.

Learning tips

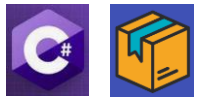
Use these strategies

- Read/watch supplied materials before classes.
- Work with supplied demos. This will help you to walk through concepts in action.
- Use workshops to work on tasks, receive feedback, and get help.
- Complete weekly tasks by recommended due dates. This will help you to build pieces for your portfolio.
- Discuss your solutions and questions with your lecturers/tutors and classmates.

Use the SIT's IT HelpHub – highly recommended

- Provides support for SIT students by answering questions, showing you where to find information, demonstrate how to solve problems.
- Is supported by tutors and volunteers.
- Check CloudDeakin for help session details

Languages/tools

Main language 

- Microsoft C#.Net (with .Net core)
- SplashKit (https://splashkit.io/)
- Weeks 1-8

Secondary language 

- Python
- Google Colaboratory
- Week 9

LET’S GET STARTED…

INTRODUCTION TO PROGRAMMING

What does programming entail?

The what...

- Develop some code that will get the computer to solve a problem or do some routine task/s
- Requires both rules and creativity

The how…

- Uses variables to store and work with data
- Uses sequence, selection, and repetition to go through computational steps

The major types…

- Procedural programming
- Object-oriented programming

Image source: https://learning.shine.com/talenteconomy/career-prospects/reasons-learn-c-sharp-language/

Procedural programming

Features

- Procedural programming breaks down tasks into step-by-step instructions for the computer to follow, like a recipe for code. It requires:
  - Variables
  - Decision and looping structures
  - Procedures and functions (sub-routines)
  - Organizing code in modules

Shortcomings

- Lack of modularity: Breaking large solutions into manageable chunks can be difficult.
- Complex change management: Changes can be cascaded everywhere in a large solution.
- Limited reusability: Difficult to reuse code in other relevant contexts.
- Data exposure: Data can be vulnerable throughout a large solution as there is no encapsulation.

INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Object-oriented programming

OOP to overcome procedural programming shortcomings

- OOP builds programs around objects that bundle data and actions, like combining ingredients and cooking instructions into self-contained recipes.
- OOP defines a real-world business/problem/system as **a set of objects**.
- Interactions between objects increases the autonomy for objects.
- Has the following four principles…
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Abstraction	Encapsulation	Inheritance	Polymorphism
Abstracting real-world entities with specific data/behaviors	Put data & behaviors together, hide unnecessary information	Entities inherit attributes and behaviors from other entities	Entities can have multiple forms and exhibit different behaviors

Sample scenario

An example OOP-based system: Melbourne zoo mgmt. system

- Melbourne zoo is home to a large number of various animals, including but not limited to pandas, koalas, gorillas, lions, and iguanas. Different animals have different dietary requirements, sleeping patterns, living conditions, and can be seen by the public at different times. Melbourne zoo has a manager who, every year, has to write a feeding, display, and enclosure cleaning schedule that the zoo-keepers use (they also have their own schedules).
- The manager has hired you to write a program that does this task...

Sample scenario

An example OOP-based system: Melbourne zoo mgmt. System (cont.)

- Data relevant to animals in the zoo
  - species, sub-species and other classifications
  - name, id
  - dietary requirements; favorite food
  - eating plan: feeding times, quantities
  - display times
  - favorite toy
  - favorite zoo keeper
- Data relevant to zoo-keepers
  - name, id
  - working shift times

Sample scenario

An example OOP-based system: Melbourne zoo mgmt. System (cont.)

- Example animals

Source of images:
https://www.zoo.org.au/melbourne/animals/red-panda

- name: Bao Bao
- species: red panda
- class: mammalian
- favorite food: bamboo

- eat(...)
- treat(...)
- sleep(...)
- make_sound(...)

- name: Gao Gao
- species: red panda
- class: mammalian
- favorite food: japanese bamboo

- eat(...)
- treat(...)
- sleep(...)
- make_sound(...)

Sample scenario

An example OOP-based system: Melbourne zoo mgmt. System (cont.)

- Procedural program (note: this example is in Python as C#.Net does not support procedural programming!)

Panda-related variables (per panda) are created separately and have no connection with each other.

```python
# keep the information of 2 pandas named Bao Bao and Gao Gao
panda_1_name = 'bao bao'
panda_1_species = 'red panda'
panda_1_class = 'mammalian'
panda_1_fav_food = 'bamboo'
panda_1_visiting_hours = 'afternoon'

panda_2_name = 'Gao Gao'
panda_2_species = 'red panda'
panda_2_class = 'mammalian'
panda_2_fav_food = 'japanese bamboo'
panda_2_visiting_hours = 'morning'
```

```python
# fetch the information of the 2 pandas
print(panda_1_name, panda_1_species, panda_1_class, panda_1_fav_food, panda_1_visiting_hours)
print(panda_2_name, panda_2_species, panda_2_class, panda_2_fav_food, panda_2_visiting_hours)

# update the information of the 2 pandas
panda_1_name = 'Bao Bao'
panda_2_fav_food = 'Japanese bamboo'
panda_1_class = 'Mammalian'

# fetch the information of the 2 pandas
print(panda_1_name, panda_1_class, panda_1_fav_food)
print(panda_2_name, panda_2_class, panda_1_fav_food) #!!
```

Sample scenario

An example OOP-based system: Melbourne zoo mgmt. System (cont.)

- OOP program

Panda-related variables and related behaviours (per panda) are created in one place and are tied with each other.

```python
class Panda:
    def __init__(self, name, species, class_, fav_food, visiting_hours):
        self.__name = name
        self.__species = species
        self.__class = class_
        self.__fav_food = fav_food
        self.__visiting_hours = visiting_hours

    def print_information(self):
        print(self.__name, self.__species, self.__class, self.__fav_food, self.__visiting_hours)
```

```python
import Panda
panda_1 = Panda.Panda('Bao Bao', 'red panda', 'Mammalian', 'bamboo', 'afternoon')
panda_1.print_information()

panda_2 = Panda.Panda('Gao Gao', 'red panda', 'Mammalian', 'Japanese bamboo', 'morning')
panda_2.print_information()
```

Ingredients of OOP

Objects

- Each object has a set of...
  - **States:** A collection of attributes and their values that define the object and, possibly, distinguish the object from other objects.
  - **Behaviors:** What the object does, i.e., the functionality that it meets, how it interacts with other objects.

- In a scenario statement, **nouns** are usually objects and state variables, **verbs** are behaviors
- An object is the fundamental building block, which can be...
  - Real, such as a light switch, student, book, or a keyboard
  - Virtual, such as an array, queue, textbox, or an avatar

Ingredients of OOP

Classes

- Are the blueprints of objects
- An object is an instance of a class, the class acts as a template for the object/s
- A class definition consists of...
  - Variables (fields)
  - Properties
  - Methods

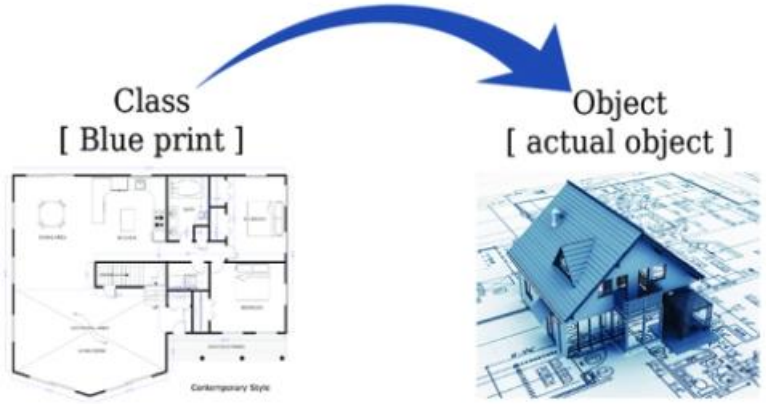


Image source: https://codestall.wordpress.com

Ingredients of OOP

Classes (cont.)

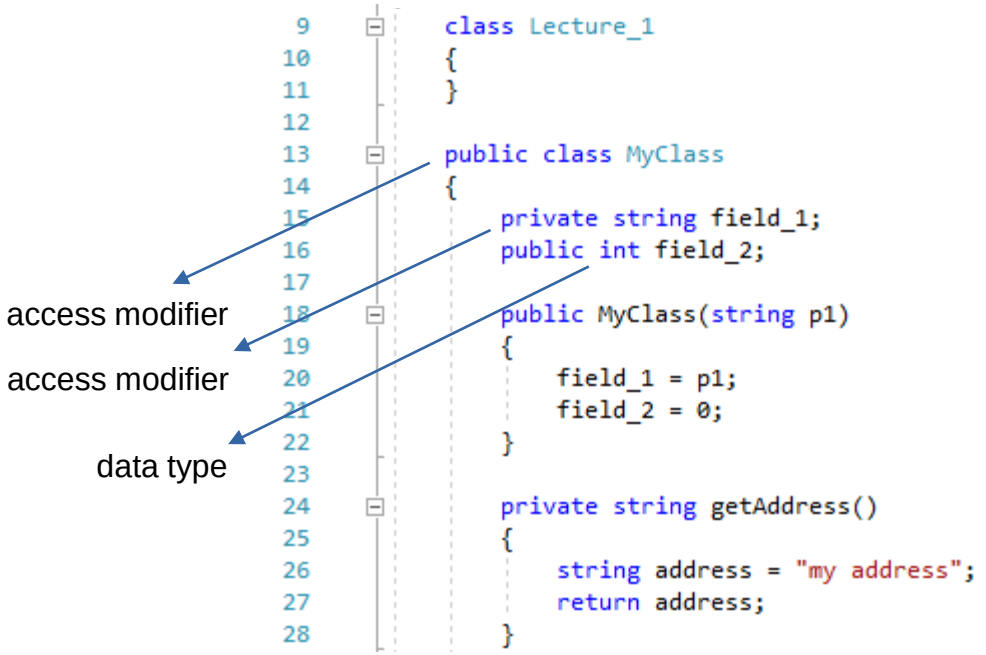
- Definition of a class and its members
  - Access modifier: public, private, and more (later)
  - The reserved word “class”

    [access_modifier] **class** class_name { … }
  - Attributes (fields, properties) and methods

    [access_modifier] **data_type** attribute_name [=value];

    [access_modifier] **return_type** method_name([parameters]) { … }

  - **Note:** Fields define each instance of a class not the class itself



Ingredients of OOP

Classes (cont.)

- Have class **constructors**, that are invoked as soon as a class is called to create an object
- Class constructors initialize the fields of a new object

```
[access_modifier] class_name([parameters])
{
 …
}
```

- **Note:** If you add parameters to a constructor, the default constructor that has no parameters will no longer be accessible (more details to come)

constructor definition with no parameters

```
13  public class MyClass
14  {
15      private string field_1;
16      public int field_2;
17
18      public MyClass()
19      {
20          field_1 = "p1";
21      }
22
23      public MyClass(string p1)
24      {
25          field_1 = p1;
26          field_2 = 0;
27      }
```

Ingredients of OOP

Classes (cont.)

- Defining or creating **objects** of a specific class

class_name object_name = **new** class_name([parameters]);

Or

class_name object_name;

object_name = **new** class_name([parameters]);

The list of these parameters must match that of one of the constructors.

```
13  public class MyClass
14  {
15      private string field_1;
16      public int field_2;
17
18      /*public MyClass()
19      {
20          field_1 = "p1";
21      }*/
22
23      public MyClass(string p1)...
28
29      private string getAddress()...
34
35      public void ReadAddress()...
39  }
40
41  public class User
42  {
43      private void method_1()
44      {
45          MyClass myInstance_1 = new MyClass();
46          MyClass myInstance_2;
47          myInstance_2 = new MyClass("p1 value");
48      }
49  }
```

```
13  public class MyClass
14  {
15      private string field_1;
16      public int field_2;
17
18      public MyClass()...
22
23      public MyClass(string p1)...
28
29      private string getAddress()...
34
35      public void ReadAddress()...
39  }
40
41  public class User
42  {
43      private void method_1()
44      {
45          MyClass myInstance_1 = new MyClass();
46          MyClass myInstance_2;
47          myInstance_2 = new MyClass("p1 value");
48      }
49  }
```

Why the error?
… = new MyClass(“value“);
Or
public MyClass()
{
}

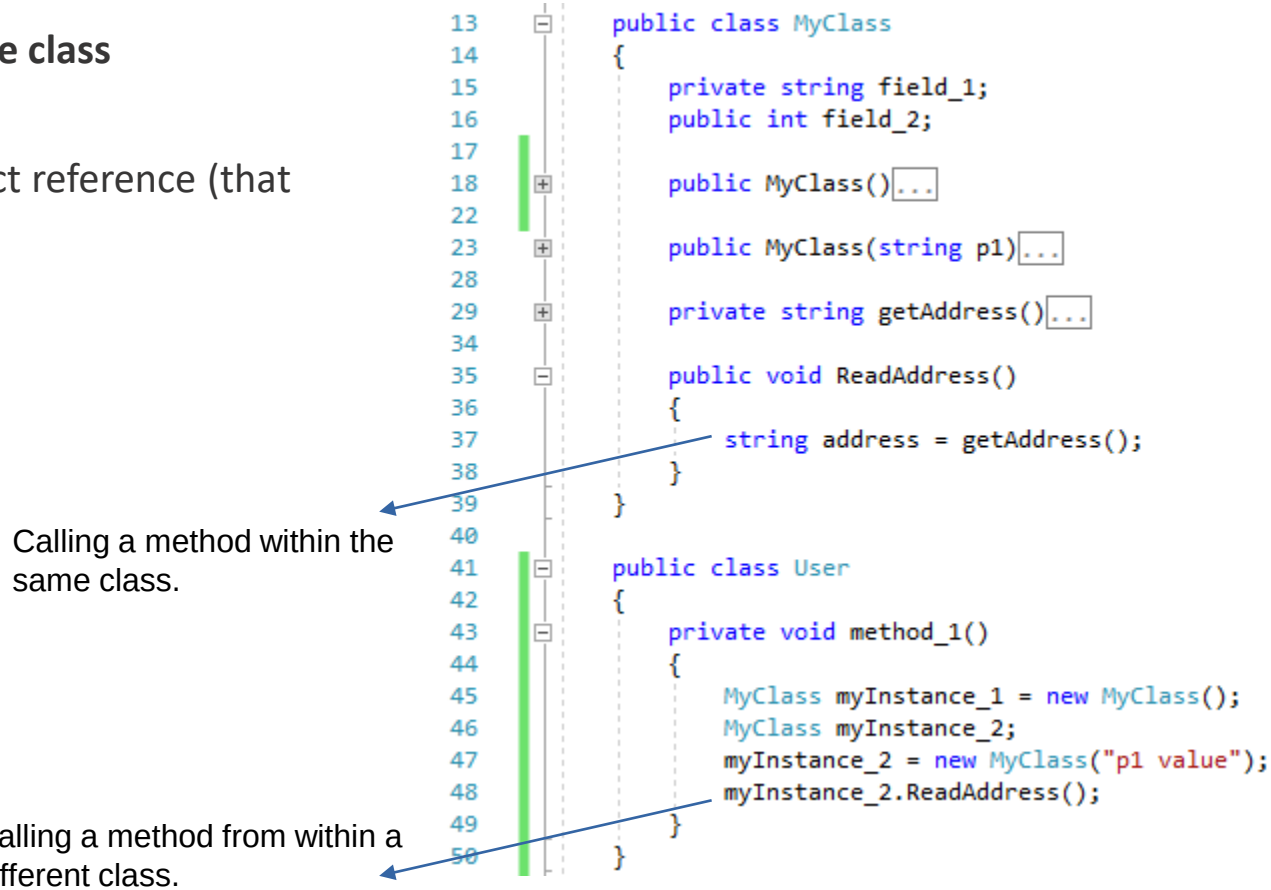
Ingredients of OOP

Classes (cont.)

- Using objects and methods
  - Calling a method from within another method in the **same class**

    method_name([parameters]);
  - Calling a method of a class in **another class** using an object reference (that has already been created)

    object_name.method_name([parameters]);




```
13  public class MyClass
14  {
15      private string field_1;
16      public int field_2;
17
18      public MyClass()...
22
23      public MyClass(string p1)...
28
29      private string getAddress()...
34
35      public void ReadAddress()
36      {
37          string address = getAddress();
38      }
39  }
40
41  public class User
42  {
43      private void method_1()
44      {
45          MyClass myInstance_1 = new MyClass();
46          MyClass myInstance_2;
47          myInstance_2 = new MyClass("p1 value");
48          myInstance_2.ReadAddress();
49      }
50  }
```

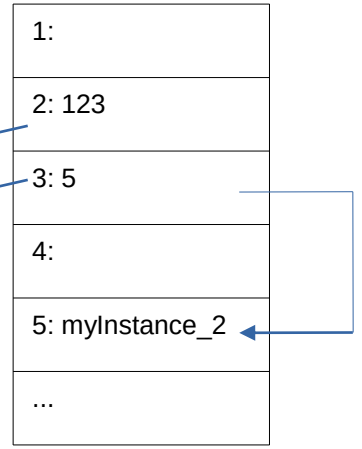
Ingredients of OOP

Classes (cont.)

- There are two types of data storage in memory
  - **Value:** The variable contains the value that has been assigned to it.
  - **Reference:** The variable contains the (memory) address of where the actual value is stored.

- When an **object** is created, the object is a **reference** type
- When a (shallow) copy of the object is made, its memory reference is copied

```
40        int student_id = 123;
41        MyClass myInstance_2;
42        myInstance_2 = new MyClass("p1 value");
```



Ingredients of OOP

Classes (cont.)

- Static fields and methods (use the **static** modifier)
  - Across classes: Static fields and methods are available with the **class name** only (not with any specific object)
  - Within the same class: Static methods only have access to other static methods
  - Examples of static methods are the read and write methods in class **Console**

```
9   class main
10  {
        0 references
11      static void Main(string[] args)
12      {
13          Console.WriteLine("This is a test message...");
14          Console.ReadKey();
15      }
16  }
17
18
19
```

```
c:\users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe
This is a test message...
```

```
13  public class MyClass
14  {
15      private string field_1;
16      public int field_2;
17
18      public MyClass()...
22
23      public MyClass(string p1)...
28
29      private string getAddress()...
34
35      public void ReadAddress()
36      {
37          string address = getAddress();
38      }
39
40      private static void staticMethod()
41      {
42          getAddress();
43      }
44  }
45
46  public class User
47  {
48      private void method_1()
49      {
50          MyClass myInstance_1 = new MyClass();
51          MyClass myInstance_2;
52          myInstance_2 = new MyClass("p1 value");
53          myInstance_2.ReadAddress();
54          myInstance_2.staticMethod();
55      }
56  }
```

The static method does not have access to non-static methods in the same class.

The static method is not accessible to any specific object.

Epilogue

KNOWLEDGE IS OF NO VALUE UNLESS YOU PUT IT INTO PRACTICE…

ANTON CHECKOV