University of Mohamed Boudiaf M'sila
Faculty of Mathematics and Computer Science
Department of Computer Science
Master 01 SIGL

Duration: 1h30 (90 Minutes) Instructor Prof. Hichem Debbi

Object Analysis and Design(ACO)
Final Exam
Sunday 19, 2025

True/False (04 points)

Write True if the statement is true, otherwise write False.

_____ Specification represents the textual dimension in UML

_____ Tagged values represent a type of constraints

_____ OCL is a part of UML.

_____ Proxy pattern is used in operating systems portability

_____ UML was the software development process part of the UML project. .

Section 2. Short Answer(08 points)

1. OCL is meant to assert objects in term of what ?
2. Define user stories. In which methodology and phase are used ?
3. What do profiles stand for ?
4. What is the main purpose of the second micro process activity
5. Consider the diagrams depicted in the following figure :

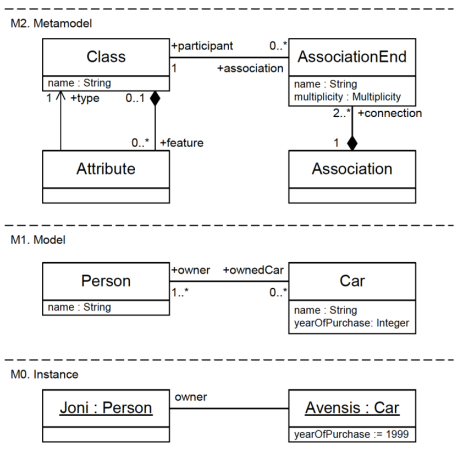

Figure 1: Class diagrams

- What do they represent ? What is the standard used ?

- Explain the entire process.
- Introduce in OCL the following constraints:
  - All cars of persons named "John" are owned before 2020.
  - The year of a car must be integer.

Section 3. Pattern Implementation(08 points)

6. In a department, there are various courses. Each course has multiple students registered. An administrator wants to view all students across courses without checking each course individually.
   Consider the following conditions:

   - Each Course has its iterator (CourseIterator), which can be used to iterate over the students of that course. The generic type T is the *Student* class in our example.
   - The Department has its own iterator (DepartmentIterator). This iterator internally leverages the iterators of the individual courses to present a seamless iteration across all students in the department, irrespective of the course they are registered in.
   - The methods, fields and implementions are depicted all togther in Figure 2.
   - An example output is depicted in Figure 3 for two courses. Please note that while *Benyounes* and *Djail* are registered in the first course, the third student *Sabir* is registered in the second course.

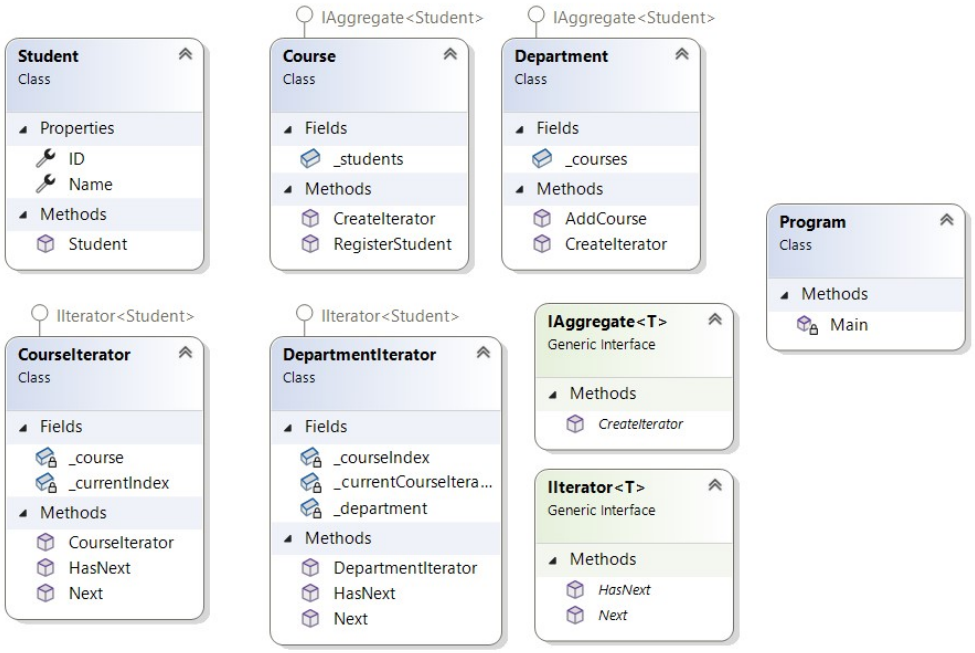

Figure 2: Class diagram

```
Name: Benyounes, ID: 1001
Name: Djail, ID: 1002
Name: Sabir, ID: 1003
```

Figure 3: Output Example

Answer Key for Exam A

True/False (04 points)

Write True if the statement is true, otherwise write False.

True Specification represents the textual dimension in UML

False Tagged values represent a type of constraints

True OCL is a part of UML.

False Proxy pattern is used in operating systems portability

False UML was the software development process part of the UML project. .

Section 2. Short Answer(08 points)

1. OCL is meant to assert objects in term of what ?

   Answer: Their methods and attributes.

2. Define user stories. In which methodology and phase are used ?

   Answer: user stories similar to use case diagram in UML. They are used in the eXtreme Programming. They can be defined in the exploration phase, and prioritized in the planning phase.

3. What do profiles stand for ?

   Answer: UML profile combines stereotypes, tagged values and constraints into one extension collection in order to customize UML models for particular domains.

4. What is the main purpose of the second micro process activity

   Answer:

5. Consider the diagrams depicted in the following figure :

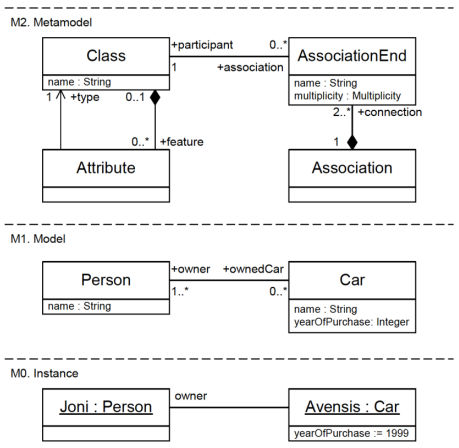

Figure 4: Class diagrams

- What do they represent ? What is the standard used ?
- Explain the entire process.
- Introduce in OCL the following constraints:
  - All cars of persons named "John" are owned before 2020.
  - The year of a car must be integer.

Answer:
- Metamodelling process
- First it comes instance model, which refers to object model, then the UML calss as it is, then the M2 metamodel represents the metamodel of the class (explanation)
- MOF Meta-Object Facility (MOF) is an Object Management Group (OMG) standard.
- OCL:
  context Owner :
  inv: self.name has type String

  context Owner:
  inv: self.name = Jhon implies self.ownedCar–¿forAll(c — c.year ¡ 2020)

Section 3. Pattern Implementation(08 points)

6. In a department, there are various courses. Each course has multiple students registered. An administrator wants to view all students across courses without checking each course individually.
   Consider the following conditions:

   - Each Course has its iterator (CourseIterator), which can be used to iterate over the students of that course. The generic type T is the *Student* class in our example.
   - The Department has its own iterator (DepartmentIterator). This iterator internally leverages the iterators of the individual courses to present a seamless iteration across all students in the department, irrespective of the course they are registered in.
   - The methods, fields and implementions are depicted all togther in Figure 2.
   - An example output is depicted in Figure 3 for two courses. Please note that while *Benyounes* and *Djail* are registered in the first course, the third student *Sabir* is registered in the second course.

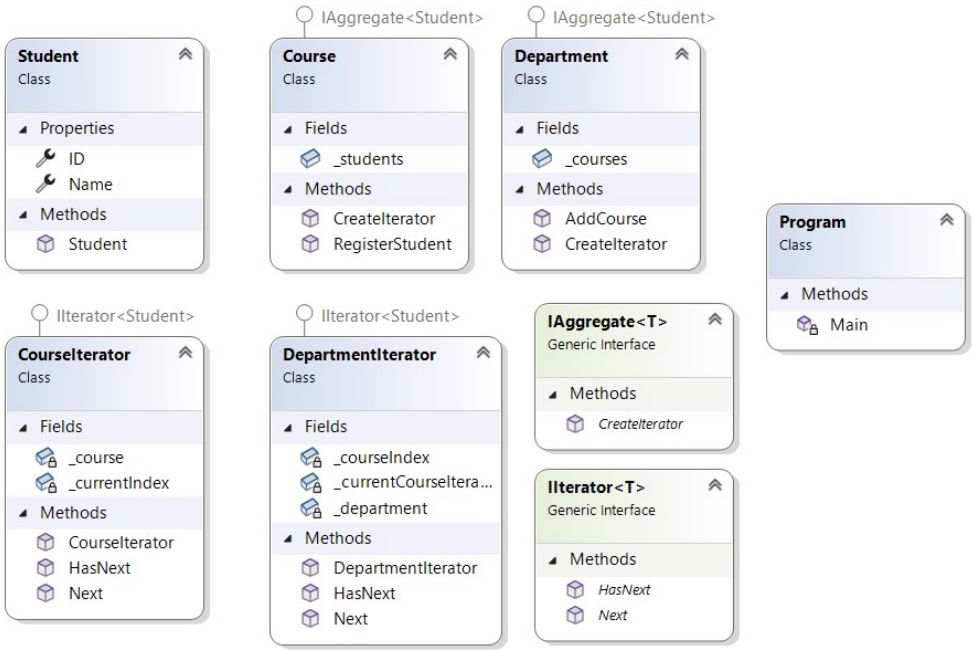

Figure 5: Class diagram

```
Name: Benyounes, ID: 1001
Name: Djail, ID: 1002
Name: Sabir, ID: 1003
```

Figure 6: Output Example

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace IteratorDesignPattern
{
    public class Student
    {
        public string Name { get; private set; }
        public string ID { get; private set; }
        public Student(string name, string id)
        {
            Name = name;
            ID = id;
        }
    }
    public interface IIterator<T>
    {
        bool HasNext();
        T Next();
    }
    public interface IAggregate<T>
    {
        IIterator<T> CreateIterator();
    }
    public class Course : IAggregate<Student>
    {
        public List<Student> _students = new List<Student>();
        public void RegisterStudent(Student student)
        {
            _students.Add(student);
        }
        public IIterator<Student> CreateIterator()
        {
            return new CourseIterator(this);
        }
    }

    public class CourseIterator : IIterator<Student>
    {
        private Course _course;
        private int _currentIndex = 0;
        public CourseIterator(Course course)
        {
            _course = course;
        }
        public bool HasNext()
        {
            return _currentIndex < _course._students.Count;
        }
        public Student Next()
        {
            return _course._students[_currentIndex++];
        }
    }

    public class Department : IAggregate<Student>
    {
        public List<Course> _courses = new List<Course>();
```

```
        public void AddCourse(Course course)
        {
            _courses.Add(course);
        }
        public IIterator<Student> CreateIterator()
        {
            return new DepartmentIterator(this);
        }
    }

    public class DepartmentIterator : IIterator<Student>
    {
        private Department _department;
        private int _courseIndex = 0;
        private IIterator<Student> _currentCourseIterator;
        public DepartmentIterator(Department department)
        {
            _department = department;
            if (_department._courses.Any())
            {
                _currentCourseIterator =
_department._courses[_courseIndex].CreateIterator();
            }
        }
        public bool HasNext()
        {
            if (_currentCourseIterator == null) return false;
            if (_currentCourseIterator.HasNext()) return true;
            // Move to the next course
            if (_courseIndex < _department._courses.Count - 1)
            {
                _currentCourseIterator =
_department._courses[++_courseIndex].CreateIterator();
                return _currentCourseIterator.HasNext();
            }
            return false;
        }
        public Student Next()
        {
            if (_currentCourseIterator != null && _currentCourseIterator.HasNext())
            {
                return _currentCourseIterator.Next();
            }
            return null;
        }
    }

    public class Program
    {
        static void Main()
        {
            var AcoCourse = new Course();
            AcoCourse.RegisterStudent(new Student("Benyounes", "1001"));
            AcoCourse.RegisterStudent(new Student("Djail", "1002"));
            var AlgoCourse = new Course();
            AlgoCourse.RegisterStudent(new Student("Sabir", "1003"));
            var department = new Department();
            department.AddCourse(AcoCourse);
```

```
            department.AddCourse(AlgoCourse);
            var iterator = department.CreateIterator();
            while (iterator.HasNext())
            {
                var student = iterator.Next();
                Console.WriteLine($"Name: {student.Name}, ID: {student.ID}");
            }

            Console.ReadLine();
        }
    }
}
```