Exercises

October 3rd 2017

This exercise sheet can be downloaded as PDF

Exercise 1 - Create an extension method

Extend the functionality of `IList<T>` to return the middle element of the collection.

This will involve creating a static class, that extends the functionality. An example can be seen below:

```
public static T GetMiddleElement<T>(this IList<T> collection)
```

Solution

```
public static T GetMiddleElement<T>(this IList<T> collection)
{
    var midIndexNumber = (int) Math.Floor(collection.Count / 2.0);
    return collection[midIndexNumber];
}
```

Exercise 2 - Create GradedCourse, BooleanCourse and Project classes

A `BooleanCourse` encapsulates a course name and a registration of passed/not passed for our sample student.

A `GradedCourse` encapsulates a course name and the grade of the student. For grading we use the Danish 7-step, numerical grades 12, 10, 7, 4, 2, 0 and -3. The grade 2 is the lowest passing grade.

In both BooleanCourse and GradedCourse you should write a method called Passed. The method is supposed to return whether our sample student passes the course.

The class `Project` aggregates two boolean courses and two graded courses. You can assume that a project is passed if at least three out of the four courses are passed. Write a method Passed in class Project which implements this passing policy.

Make a project with four courses, and try out your solution.

Solution

This solution only contains the courses and project code.

```
public class BooleanCourse
{
    public string CourseName { get; set; }
    private bool _grade;

    public bool Passed()
    {
        return _grade;
    }

    public void SetGrade(bool grade)
    {
        _grade = grade;
    }
}

public class GradedCourse
{
    private enum Grade
    {
        NotGraded, MinusThree = -3, Zero = 0,
        Two = 2, Four = 4, Seven = 7,
        Ten = 10, Twelve = 12
    }

    public string CourseName { get; set; }
    private Grade _grade = Grade.NotGraded;

    public bool Passed()
    {
        return _grade != Grade.MinusThree && _grade != Grade.Zero;
    }

    public void SetGrade(Grade grade)
    {
        _grade = grade;
    }
}

public class Project
{
    public BooleanCourse[] BooleanCourses = new BooleanCourse[2];
    public GradedCourse[] GradedCourses = new GradedCourse[2];

    public bool Passed()
    {
        var passedCourses = BooleanCourses.Count(x => x.Passed());
        passedCourses += GradedCourses.Count(x => x.Passed());

        return passedCourses >= 3;
    }
}
```

Exercise 3 - Rewrite GradedCourse and BooleanCourse to use inheritance

Create a base class called Course which BooleanCourse and GradedCourse inherits from.

Move common fields to the Course class.

Change the Project class to accept Course classes and adapt the methods Passed().

If you can see a better way to use inheritance here, implement it that way. You decide between viable solutions.

```
public class GradedCourse : Course {
    ...
}
```

Try out your solution.

Solution

This solution only contains the courses and project code.

```
public abstract class Course
{
    public string CourseName { get; set; }

    public abstract bool Passed();
}

public class BooleanCourse : Course
{
    private bool _grade;

    public override bool Passed()
    {
        return _grade;
    }

    public void SetGrade(bool grade)
    {
        _grade = grade;
    }
}

public class GradedCourse : Course
{
    private enum Grade
    {
        NotGraded, MinusThree = -3, Zero = 0,
        Two = 2, Four = 4, Seven = 7,
        Ten = 10, Twelve = 12
    }

    private Grade _grade = Grade.NotGraded;

    public override bool Passed()
    {
        return _grade != Grade.MinusThree && _grade != Grade.Zero;
    }

    public void SetGrade(Grade grade)
    {
        _grade = grade;
    }
}

public class Project
{
    public Course[] Courses = new Course[4];

    public bool Passed()
    {
        return Courses.Count(course => course.Passed()) >= 3;
    }
}
```