

C# OO Programmeren

LES 7

JOHAN DONNÉ

Custom Data Types revisited:

- Enum types
- Classes
- Structs

Classes

Classes

```
public class Participant: Person, IEquatable<Participant>,
                               IComparable<Participant>
{
    private int registrationNumber;

    public String Name { get; }
    public List<int> Preferences { get; }

    public Participant(string name, List<int> preferences)
    {
        this.Name = name;
        this.Preferences = preferences;
    }
}
```

Velden, properties, constructoren, methodes,
base class, implemented interfaces

Class inheritance

```
public class SuperClass
{
    public int dummy { get; set; }
    ...
}

public class SubClass : SuperClass
{
    public string AnotherDummy { get; set; }
    ...
}
```

‘SubClass’ erft alle publieke en protected velden, properties en methodes van ‘SuperClass’

‘SubClass’ voegt zijn eigen velden, properties, methodes toe.

Private velden van ‘SuperClass’ zijn wel aanwezig in ‘SubClass’ maar niet zichtbaar.

Subclass constructors


Opgelet: een subklasse erft niet de constructoren van de superklasse!

⇒ Subklasse moet eigen constructor(en) hebben.

Maar: vaak moet de initialisatie uit de constructoren van de superklasse ook gebeuren.

- initialisatiecode dupliceren is ongewenst
- als sourcecode van superklasse niet beschikbaar is, weet je ook niet welke initialisatie er precies gebeurt.

Subclass constructors



```
public class Person
{
    private string name;
    private int age;

    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

```
public class Student: Person
{
    private string course;

    public Student(string name, int age, string course) : base(name, age)
    {
        this.course = course;
    }
}
```

Subclass constructors

```
public class Student: Person
{
    private string course;

    public Student(string name, int age, string course) : base(name, age)
    {
        this.course = course;
    }
}
```

```
Student student = new Student("Helen", 18, "Math");
```

- Er wordt een student object aangemaakt,
- De constructor van 'Person' wordt opgeroepen ("base(name, age)").
- De constructor van 'Student' wordt uitgevoerd.

⇒ Is ook mogelijk bij constructor overloading...

Overriding virtual methods

Behalve voor het toevoegen van extra velden/properties, kan je inheritance ook gebruiken om het 'gedrag' van een klasse te wijzigen.

⇒ Methodes vervangen door alternatieve versies ervan.

```
public class Person
{
    private string name;
    private int age;
    public virtual string GetName()
    {
        return this.name ;
    }
}
```

```
public class Student: Person
{
    private string course;

    public override string GetName()
    {
        return base.GetName() + " (student)";
    }
}
```

Overriding virtual methods

Behalve voor het toevoegen van extra velden/properties, kan je inheritance ook gebruiken om het 'gedrag' van een klasse te wijzigen.

⇒ Methodes vervangen door alternatieve versies ervan.

```
public class Person
{
    private string name;
    private int age;
    public virtual string GetName()
    {
        return this.name ;
    }
}
```

```
public class Student: Person
{
    private string course;

    public override string GetName()
    {
        return base.GetName() + " (student)";
    }
}
```

```
Person p1 = new Person("Helen", 10);
Console.WriteLine(p1.GetName());
Student s1 = new Student("Jim", 18, "Math");
Console.WriteLine(s1.GetName());

Person p2 = s1;
Console.WriteLine(p2.GetName());
```



```
Helen
Jim (student)
Jim (student)
```

'Overriding' versus 'Hiding'

Behalve voor het toevoegen van extra velden/properties, kan je inheritance ook gebruiken om het 'gedrag' van een klasse te wijzigen.

⇒ Methodes vervangen door alternatieve versies ervan.

```
public class Person
{
    private string name;
    private int age;
    public virtual string GetName()
    {
        return this.name ;
    }
}
```

```
public class Student: Person
{
    private string course;
    new public string GetName()
    {
        return base.GetName() + " (student)";
    }
}
```

```
Person p1 = new Person("Helen", 10);
Console.WriteLine(p1.GetName());
Student s1 = new Student("Jim", 18, "Math");
Console.WriteLine(s1.GetName());

Person p2 = s1;
Console.WriteLine(p2.GetName());
```



```
Helen
Jim (student)
Jim
```

‘Overriding’ versus ‘Hiding’

	Overriding	Hiding
methode declaratie	‘virtual’ – ‘override’	‘virtual’ – ‘new’
oproep via subklasse	methode uit subklasse	methode uit subklasse
Oproep via superklasse	methode uit subklasse	methode uit superklasse

Sealed classes, sealed members

```
public sealed class Person
{
    ...
}
```

Sealed class kan niet meer als base klas gebruikt worden

```
public sealed override string GetName()
{
    return base.GetName() + " (student)";
}
```

In verder afgeleide klassen kan geen override meer gebeuren van de sealed method.

Abstract classes, abstract members

```
public abstract class Shape
{
    public Point Position { get; set; }
    public abstract double Area { get; }
}
```

Abstracte klasse kan enkel als base class gebruikt worden, kan niet zelf geïntantieerd worden.

Elke afgeleide klasse **moet** een override doen van elke abstracte methode.

Een abstracte klasse kan ook gewone properties/methodes bevatten (cfr. 'Position')

```
public class Square : Shape
{
    public double Side { get; set; }
    public override double Area
    {
        get
        {
            return Side * Side;
        }
    }
}
```

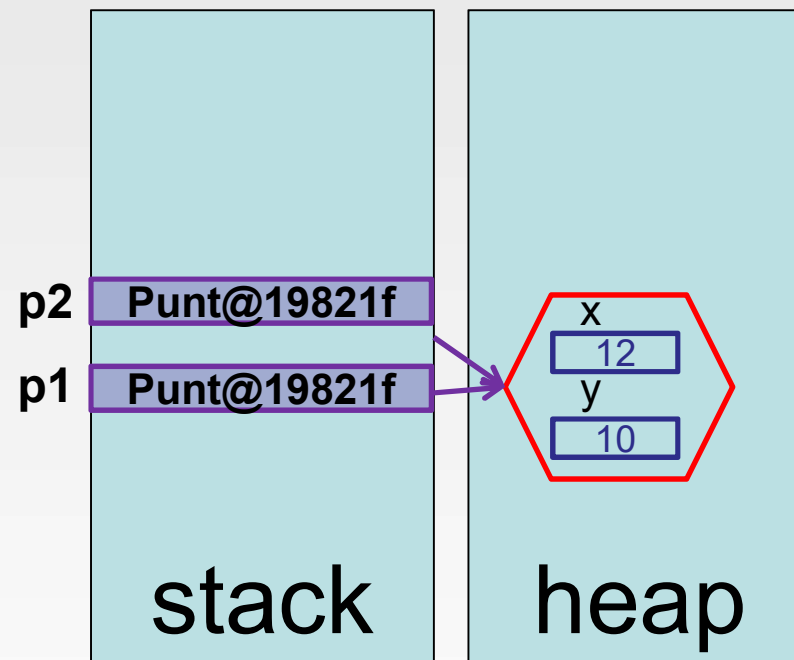
Structs

Class

Een klasse is een custom type dat je zelf definieert.

Een klasse is een 'reference' type:

```
Punt p1 = new Punt(12,10);  
Punt p2 = p1;
```

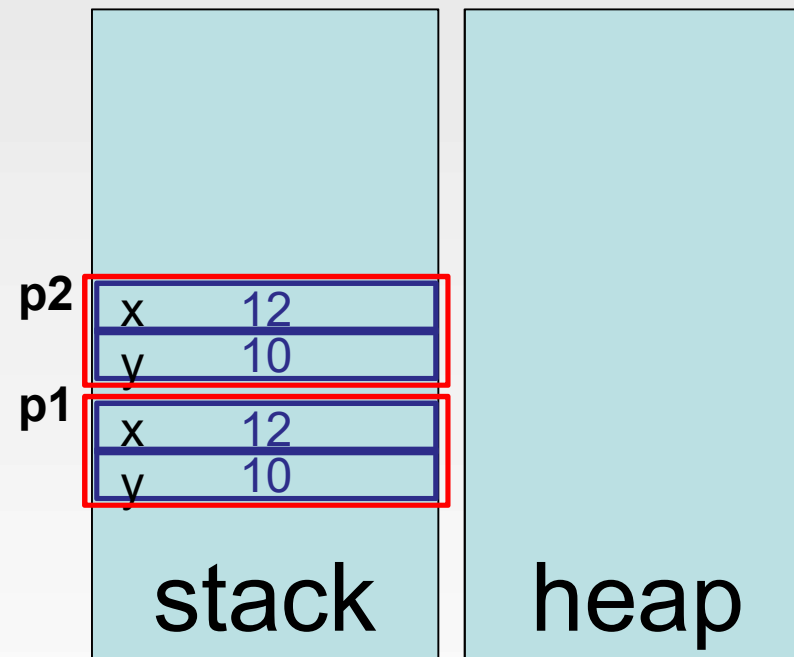


Struct

Een struct is gelijkaardige aan een klasse (dus ook een custom type dat je zelf definieert).

Een struct is een 'value' type:

```
Punt p1 = new Punt(12,10);  
Punt p2 = p1;
```



Struct

Declaratie is gelijkaardig als bij klasse.

Ook velden, properties, methodes, constructor

Geen inheritance, abstract, sealed... (is impliciet 'sealed')

Kan wel interfaces implementeren

```
public struct Point2D
{
    public Point2D(int x, int y)
    {
        X = x; Y= y;
    }
    public int X { get; set; }
    public int Y { get; set; }
}
```

Typisch:

- wanneer data belangrijker is dan 'gedrag'
- Wanneer een 'value' type handiger is dan een 'reference' type (cloning problematiek)

Struct

Opgelet bij Arrays, Collections van structs:

```
var list = new List<Point2D> { new Point2D(0,0), new Point2D(10,10) };  
  
var point = list[1];  
point.X = 5;  
  
list[0].X = 5;
```

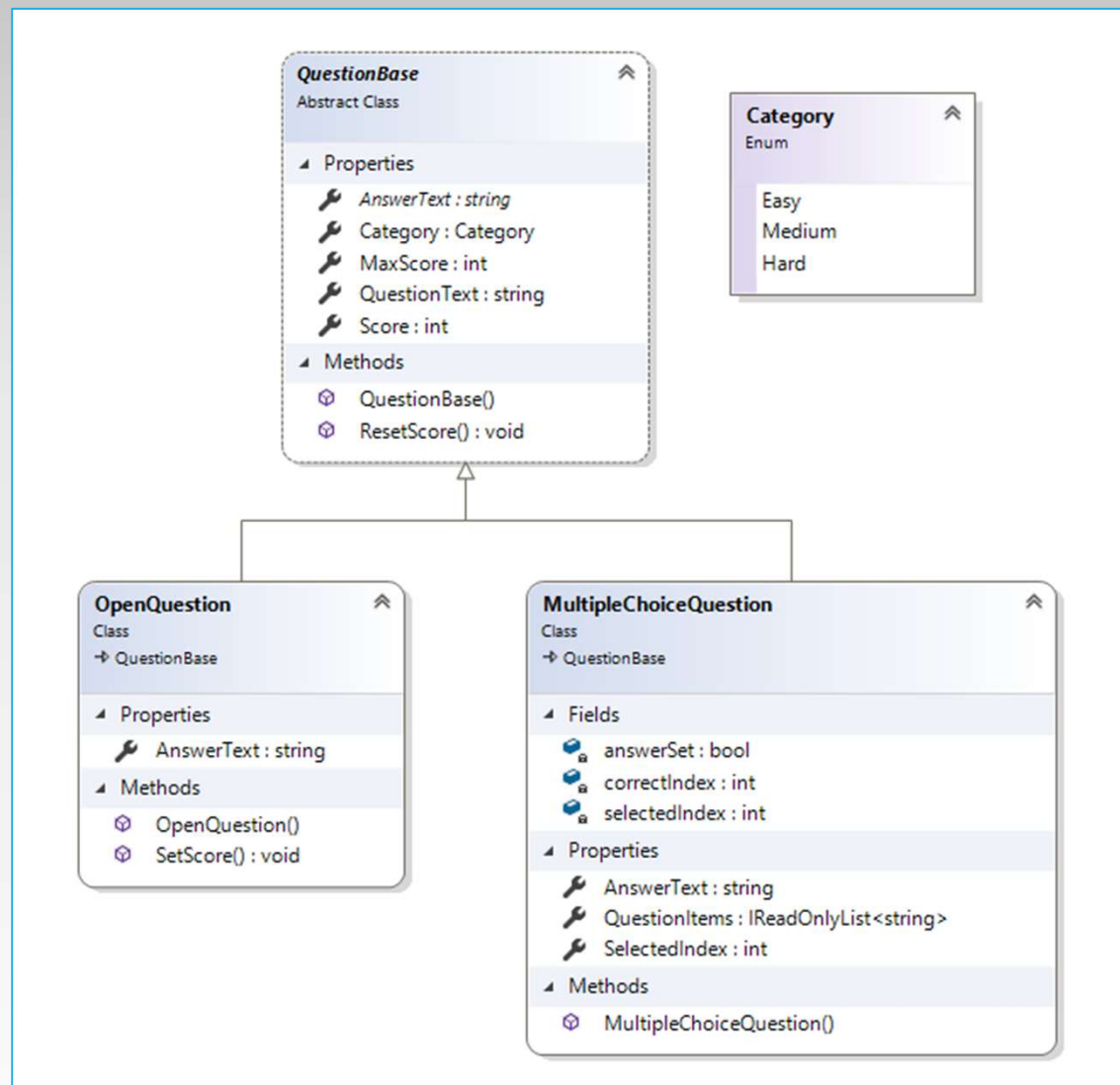
Cannot modify the return value of 'List<Point2D>.this[int]' because it is not a variable

- `point.X = 5;` werkt, maar wijzigt niets aan de lijst.
- `list[0]` geeft een 'waarde' terug en geen 'variabele'.
- `list[0] = new Point2D(5, list[0].Y);` lukt wel.

Computer-examensysteem

- ⇒ Klassehiërarchie voor verschillende soorten vragen
- ⇒ Combinatie met lagenmodel
- ⇒ Data laag: Serialisatie en deserialisatie van polymorfe collections
- ⇒ Presentatielaag: klassehiërarchie voor presentatie van verschillende soorten vragen

Entities



QuestionBase

```
public abstract class QuestionBase
{
    // 'Score' & 'AnswerText' should not be serialized to the questionFile
    // since they are not ment to be filled in when reading from file.

    // example of a 'readonly' property: can only be initialized from the constructor
    public Category Category { get; }
    public int MaxScore { get; }

    [JsonIgnore]
    public virtual int Score { get; protected set; }

    // example of a 'readonly' property: can only be initialized from the constructor
    public string QuestionText { get; }

    [JsonIgnore]
    public abstract string AnswerText { get; set; }

    public QuestionBase(Category category, string questionText, int maxScore)
    {
        Category = category;
        QuestionText = questionText;
        MaxScore = maxScore;
    }

    public virtual void ResetScore()
    {
        Score = 0;
    }
}
```

OpenQuestion

```
public class OpenQuestion : QuestionBase
{
    public override string AnswerText { get; set; }

    public OpenQuestion(Category category, string questionText, int maxScore)
        : base(category, questionText, maxScore)
    {
    }

    public void SetScore(int score)
    {
        base.Score = score;
    }
}
```

MultipleChoiceQuestion

```
public class MultipleChoiceQuestion : QuestionBase
{
    // correctIndex should be private, but read from questionfile, so make it a Jsonproperty
    [JsonProperty]
    private readonly int correctIndex;

    private int selectedIndex;
    private bool answerSet = false;

    public IReadOnlyList<string> QuestionItems { get; }

    public override string AnswerText
    {
        get => answerSet ? QuestionItems[SelectedIndex] : string.Empty;
        set => throw new InvalidOperationException("AnswerText is ReadOnly");
    }

    [JsonIgnore]
    public int SelectedIndex
    {
        get => selectedIndex;
        set
        {
            answerSet = true;
            selectedIndex = value;
            base.Score = SelectedIndex == correctIndex ? MaxScore : 0;
        }
    }

    public MultipleChoiceQuestion(Category category, string questionText, int maxScore,
        List<string> questionItems, int correctIndex) : base(category, questionText, maxScore)
    {
        QuestionItems = questionItems.AsReadOnly();
        this.correctIndex = correctIndex;
    }

    public override void ResetScore()
    {
        base.ResetScore();
        answerSet = false;
        selectedIndex = 0;
    }
}
```


Polymorfe (de)serialisatie

```
private void ReadQuestions()
{
    var formatter = new JsonSerializer { TypeNameHandling = TypeNameHandling.Auto };
    using (var file = new StreamReader(questionFilename))
    {
        questionPool = (Dictionary<Category, List<QuestionBase>>)formatter.Deserialize
            (file, typeof(Dictionary<Category, List<QuestionBase>>));
    }
    readOnlyQuestionPool = new ReadOnlyDictionary<Category, List<QuestionBase>>(questionPool);
}

private void StoreQuestions()
{
    var formatter = new JsonSerializer { TypeNameHandling = TypeNameHandling.Auto }; ;
    using (var file = File.CreateText(questionFilename))
    {
        formatter.Serialize(file, questionPool);
        file.Flush();
    }
}
```

Logische laag: assesment

```
public class Assessment : IAssessment
{
    private readonly List<QuestionBase> questions;

    public IReadOnlyList<QuestionBase> Questions => questions.AsReadOnly();
    public int MaxScore { get; }
    public int ActualScore
    {
        get
        {
            int score = 0;
            foreach (var question in questions) score += question.Score;
            return score;
        }
    }

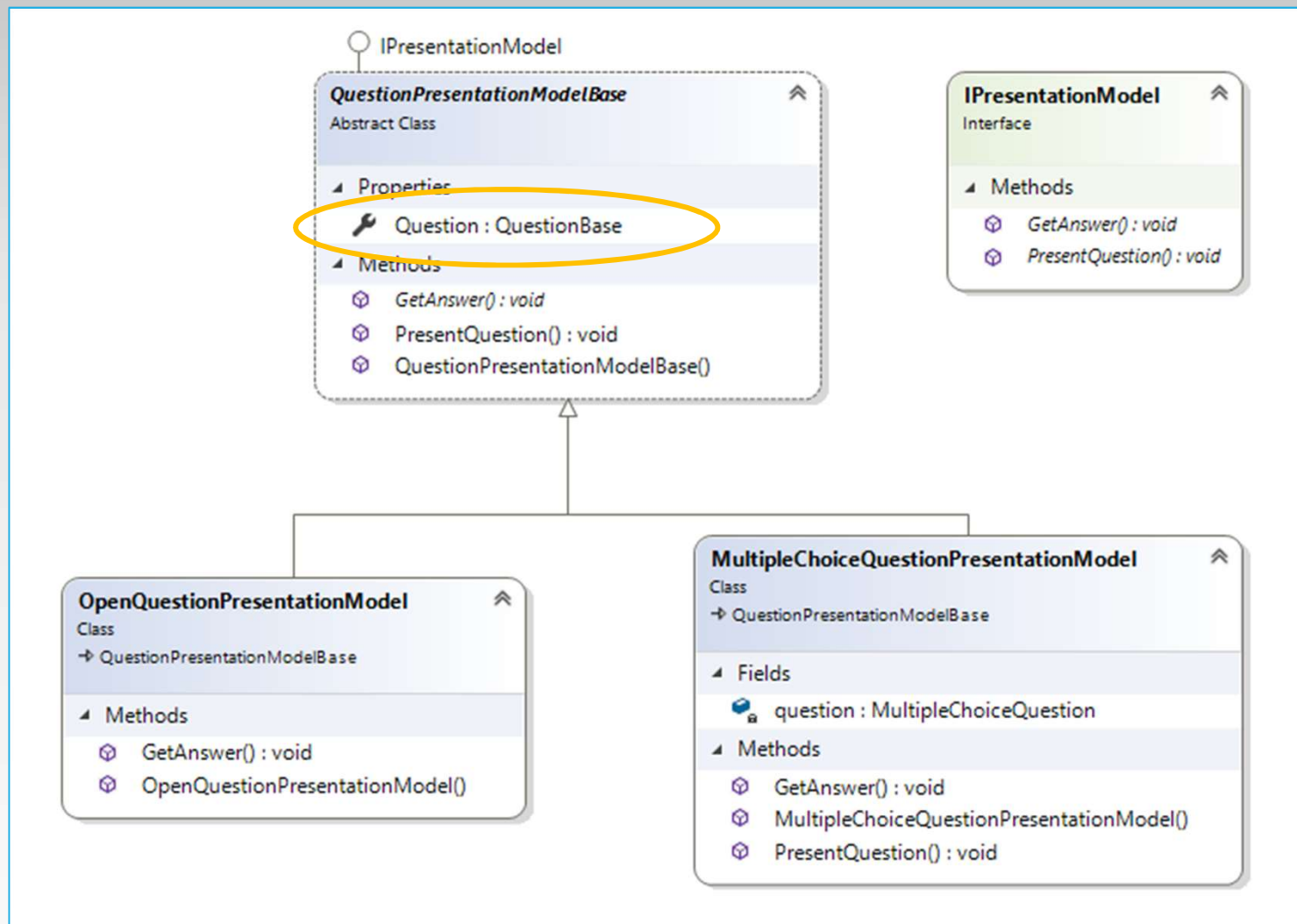
    public Assessment(IQuestionStore questionStore)
    {
        // create assessment questionList by choosing a number of questions from each category randomly
        // for the demo: just select first question for each category

        questions = new List<QuestionBase>();
        foreach (var category in questionStore.QuestionPool.Keys)
        {
            var question = questionStore.QuestionPool[category].First();
            questions.Add(question);
        }

        MaxScore = 0;
        foreach (var question in questions) MaxScore += question.MaxScore;
    }

    public void Save()
    { ... }
}
```

Presentatielaag - consoletoepassing



Presentatie: aanmaken lijst presentatiemodellen

```
public class AssessmentConsoleUi
{
    private readonly IAssessment assessment;
    private List<IPresentationModel> questionList = new List<IPresentationModel>();

    public AssessmentConsoleUi(IAssessment assessment)
    {
        this.assessment = assessment;
        ConstructQuestionPresentationList();
    }

    private void ConstructQuestionPresentationList()
    {
        foreach (var question in assessment.Questions)
        {
            switch (question)
            {
                case OpenQuestion q:
                {
                    questionList.Add(new OpenQuestionPresentationModel(q));
                    break;
                }

                case MultipleChoiceQuestion q:
                {
                    questionList.Add(new MultipleChoiceQuestionPresentationModel(q));
                    break;
                }

                default: break;
            }
        }
    }
}
```

