Mapping to a class-based object oriented model

course: Information Modelling

Translating entity types to classes

course: Information Modelling

Quick recap…

ENTITY

*An **entity** is a "thing" which can be distinctly identified.*
A specific person, company, or event is an example of an entity.

Peter Pin-Shan Chen. 1976.

Quick recap...

ENTITY SET

It is a **set (or collection) of entities** or **instances of an entity type** which share the similar properties or attributes.

Quick recap...

ENTITY TYPE

It is a **concept** for defining the entities which share the similar properties or attributes.

Ann and Jim are Member entities in VGC's UoD

- In OO parlance they are also called objects
- In this course we will use C# as a class-based OO language
- Leverage our existing C# knowledge

Mapping an entity type to a class

```
namespace Domain;

public class Member(int memberNo, string name, string firstName)
{

    // TODO Add member-specific attributes

}
```

Mapping an entity type to a class

ENTITY TYPE

```
namespace Domain;

public class Member(int memberNo, string name, string firstName)
{

    // TODO Add member-specific attributes

}
```

Creating entities and entity sets

ENTITIES

ENTITY SET

```
using Domain;

Member c = new(1, "Theben Tervile", "Corneel");
Member m = new(2, "De Groef", "Machteld");
Member f = new(3, "Vlummens", "Frédéric");

IList<Member> members = [ c, m, f ];

Console.WriteLine(members.Count);
```

Translating attributes to classes

course: Information Modelling

Attributes: basic pattern

```
public class Member
{

    private final int _membershipNumber;
    private final string _lastName;
    private final string _firstName;

}
```

Accessors (getters) and mutators (setters) are to be defined as needed

(cfr. OO classes)

Attributes: basic pattern

```
public class Member(int memberNo, string name, string firstName)
{

    private int _memberNo = memberNo;

    private string _name = name;

    private string _firstName = firstName;

    ...
}
```

Mandatory attributes should be present upon creation of member

Attributes: basic pattern

```
public Member(int memberNo, string name, string firstName, string title)
{
    MemberNo = memberNo;
    Name = name;
    FirstName = firstName;
    Title = title;
}

public int MemberNo
{
    get => _memberNo;
    private set
    {
        ArgumentOutOfRangeException.ThrowIfLessThan(value, 1);
        _memberNo = value;
    }
}

public string Name
{
    get => _name;
    private set => _name = value ??  throw new ArgumentNullException(nameof(Name));
}
```

Mandatory attributes should be present upon creation of member ➔ even better

Translating optional attributes to classes

course: Information Modelling

Translating optional attributes to classes

```
private string? _title;

public Member(int memberNo, string name, string firstName, string? title = null)
{
    MemberNo = memberNo;
    Name = name;
    FirstName = firstName;
    Title = title;
}

public string? Title
{
    get => _title;
    private set => _title = value;
}
```

Translating derived attributes to classes

course: Information Modelling

Translating derived attributes to classes

- Remember: choosing information that changes continuously is not a good choice for attributes
- Example: **age**
- Better: use something that doesn't change, such as **date of birth**

Calculating the age

```
private DateOnly _dateOfBirth;

public Member(int memberNo, string name, string firstName, DateOnly dateOfBirth)
{
    MemberNo = memberNo;
    Name = name;
    FirstName = firstName;
    DateOfBirth = _dateOfBirth;
}

public int Age
{
    get
    {
        DateOnly today = DateOnly.FromDateTime(DateTime.Today);
        int age = today.Year - DateOfBirth.Year;

        if (today < DateOfBirth.AddYears(age))
        {
            age--;
        }

        return age;
    }
}
```

Calculating the age

```
private DateOnly _dateOfBirth;

public Member(int memberNo, string name, string firstName, DateOnly dateOfBirth)
{
    MemberNo = memberNo;
    Name = name;
    FirstName = firstName;
    DateOfBirth = _dateOfBirth;
}

public int Age
{
    get
    {
        DateOnly today = DateOnly.FromDateTime(DateTime.Today);
        int age = today.Year - DateOfBirth.Year;

        if (today < DateOfBirth.AddYears(age))
        {
            age--;
        }

        return age;
    }
}
```

Dynamically derive the (changing) age from the date of birth (which is constant in time)

Translating multivalued attributes to classes

course: Information Modelling

Translating multivalued attributes to classes

- Example: multiple telephone numbers
- 050/123456; 0475/654321; ...

Translating multivalued attributes to classes

```
public class Member
{
    private ISet<string> _phoneNumbers = new HashSet<string>();

    public void AddPhoneNumber(string phoneNumber)
    {
        _phoneNumbers.Add(phoneNumber);
    }

    public IReadOnlySet<string> PhoneNumbers => _phoneNumbers.ToHashSet();
}
```

Translating multivalued attributes to classes

```
public class Member
{
    private ISet<string> _phoneNumbers = new HashSet<string>();

    public void AddPhoneNumber(string phoneNumber)
    {
        _phoneNumbers.Add(phoneNumber);
    }

    public IReadOnlySet<string> PhoneNumbers => _phoneNumbers.ToHashSet();
}
```

Translating multivalued attributes to classes

```
public class Member
{
    private ISet<PhoneNumber> _phoneNumbers = new HashSet<PhoneNumber>();

    public void AddPhoneNumber(PhoneNumber phoneNumber)
    {
        _phoneNumbers.Add(phoneNumber);
    }

    public IReadOnlySet<PhoneNumber> PhoneNumbers => _phoneNumbers.ToHashSet();
}
```

Not limited to built-in types such as ints and strings
You can also create new entities (e.g. PhoneNumber) and keep these as a collection (set/list/map/...).

Translating composite attributes to classes

course: Information Modelling

Translating composite attributes to classes

- No out of the box solution
- If the composite attributes are isolated in a separate entity, you can build a separate class

Translating composite attributes to classes

```
public class PhoneNumber(string countryCode, string prefix, string number)
{
    public string CountryCode => countryCode;
    public string Prefix => prefix;
    public string Number => number;

    public override string ToString() => $"+{countryCode}{prefix}{number}";
}
```

Translating identity to classes

course: Information Modelling

Translating identity to classes

- Each member has a unique membership number
- How can we tell the CLR when two members are compared whether they are equal?

Translating identity to classes

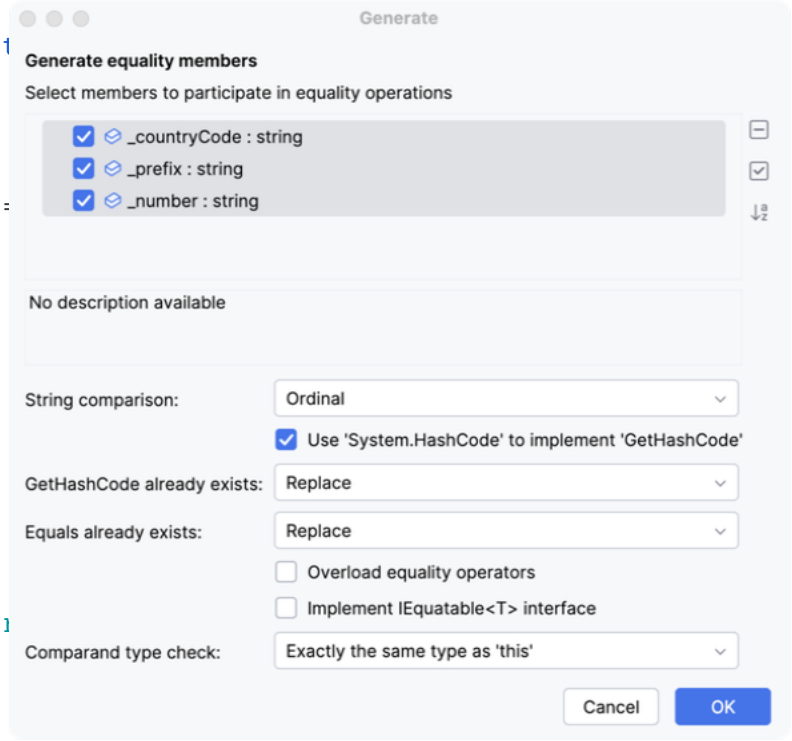
- Each member has a unique membership number
- How can we tell the CLR when two members are compared whether they are equal?
- Solution: implement the Equals and GetHashCode methods

Translating identity to classes

```
public class PhoneNumber(string countryCode, string prefix, string number)
{

    protected bool Equals(PhoneNumber other)
    {
        return _countryCode == other._countryCode && _prefix == other._prefix && _number == other._number;
    }

    public override bool Equals(object? obj)
    {
        if (obj is null) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != GetType()) return false;
        return Equals((PhoneNumber)obj);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(_countryCode, _prefix, _number);
    }
}
```

Translating identity to classes

```
public class PhoneNumber(string countryCode, string prefix, st
{

    protected bool Equals(PhoneNumber other)
    {
        return _countryCode == other._countryCode && _prefix =
    }

    public override bool Equals(object? obj)
    {
        if (obj is null) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != GetType()) return false;
        return Equals((PhoneNumber)obj);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(_countryCode, _prefix, _numbe
    }
}
```



Translating weak entities to classes

course: Information Modelling

DEFINITION

WEAK ENTITY

If the existence of an entity depends on the existence of another entity, we say the entity is a weak entity.

Peter Pin-Shan Chen. 1976.

Weak entities

- VGC:

Translating weak entities to classes

```
public class Subscription(Member member, DateOnly startDate, DateOnly endDate, decimal fee)
{
    public Member Member => member;
    public DateOnly StartDate => startDate;
    public DateOnly EndDate => endDate;
    public decimal Fee => fee;

    private bool Equals(Subscription other)
    {
        return Member.Equals(other.Member) && StartDate.Equals(other.StartDate);
    }

    public override bool Equals(object? obj)
    {
        if (obj is null) return false;
        if (ReferenceEquals(this, obj)) return true;
        return obj.GetType() == GetType() && Equals((Subscription)obj);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(Member, StartDate);
    }
}
```

Translating weak entities to classes

```
public class Subscription(Member member, DateOnly startDate, DateOnly endDate, decimal fee)
{
    public Member Member => member;
    public DateOnly StartDate => startDate;
    public DateOnly EndDate => endDate;
    public decimal Fee => fee;

    private bool Equals(Subscription other)
    {
        return Member.Equals(other.Member) && StartDate.Equals(other.StartDate);
    }

    public override bool Equals(object? obj)
    {
        if (obj is null) return false;
        if (ReferenceEquals(this, obj)) return true;
        return obj.GetType() == GetType() && Equals((Subscription)obj);
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(Member, StartDate);
    }
}
```

Translating relationships to classes

course: Information Modelling

Different types of relationships

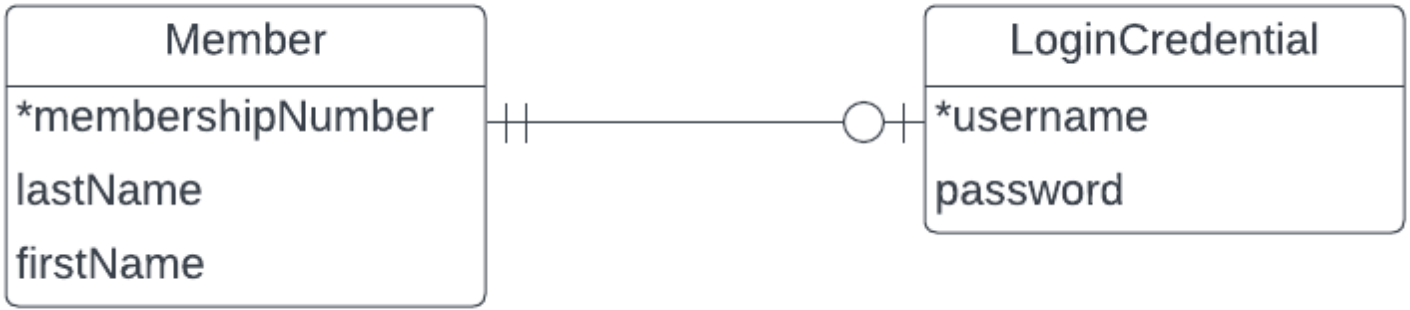
- One-to-one
- One-to-many
- Many-to-many

Remember...

PARTICIPATION and CARDINALITY

The **participation of a relationship** defines the **minimum** number of related entities on a relationship (0 or 1), the **cardinality of a relationship** defines the **maximum** number of related entities on a relationship (1 or more).

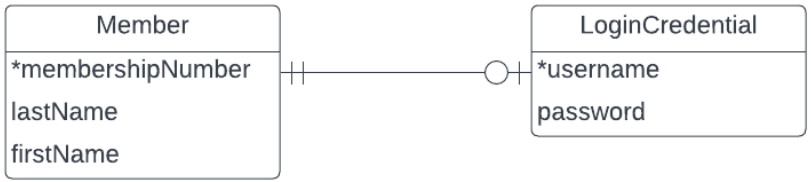
One-to-one



- All members are defined
- Some members may choose to use the self-service website and therefore create a username/password combo

Composition and one-to-one

```
public class Member
{
    private readonly IDictionary<DateOnly, Subscription> _subscriptions = new Dictionary<DateOnly, Subscription>();
    private readonly IList<BoardGame> _reservedBoardGames = new List<BoardGame>();
    private readonly LoginCredential? _loginCredential;

    public Member(int memberNo, string name, string firstName, LoginCredential? loginCredential = null)
    {
        ArgumentOutOfRangeException.ThrowIfLessThan(memberNo, 1);

        MemberNo = memberNo;

        Name = name ?? throw new ArgumentNullException(nameof(name));

        FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));

        _loginCredential = loginCredential;
    }
```



One-to-many

- A Member has one or more subscriptions
- Here, Subscription is a weak entity ➔ identified by memberNo and startDate

One-to-many

```
public class Subscription(Member member, DateOnly startDate,
    DateOnly endDate, decimal fee)
{
    public Member Member => member;
    public DateOnly StartDate => startDate;
    public DateOnly EndDate => endDate;
    public decimal Fee => fee;
}
```

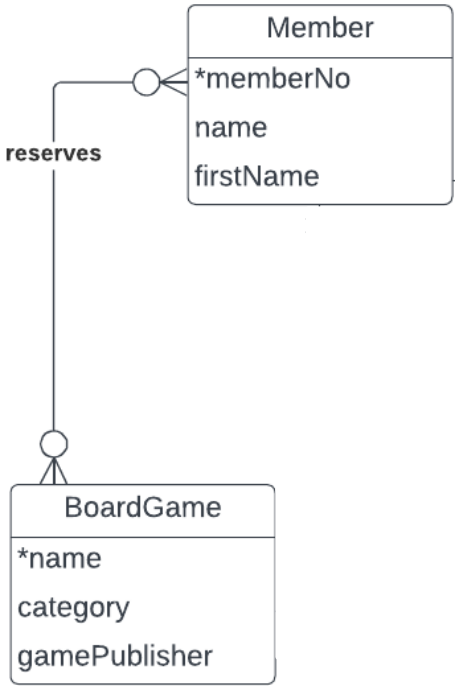
```
public class Member
{

    public void Subscribe(DateOnly startDate, int months, decimal fee)
    {
        try
        {
            Subscription subscription = new(this, startDate,
startDate.AddMonths(months).AddDays(-1), fee);
            _subscriptions.Add(startDate, subscription);
        }
        catch (ArgumentException)
        {
            throw new VgcException($"Subscription for {startDate} already exists");
        }
    }

    public IDictionary<DateOnly, Subscription> Subscriptions => _subscriptions.AsReadOnly();

    public void AddReservation(BoardGame game)
    {
        _reservedBoardGames.Add(game);
    }

}
```

One-to-many

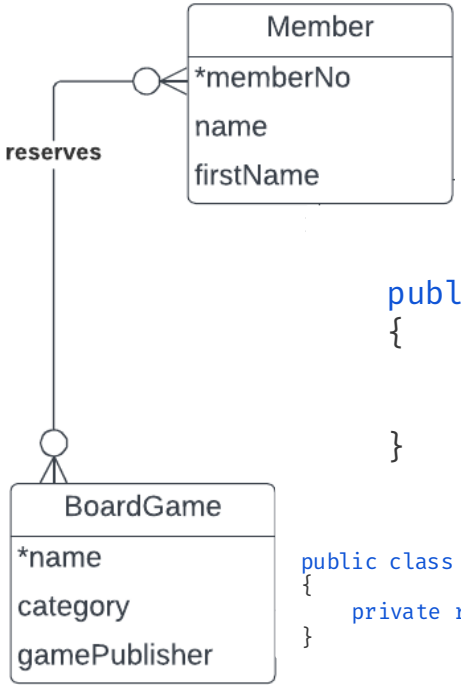
```
public class Subscription(Member member, DateOnly startDate,
    DateOnly endDate, decimal fee)
{
    public Member Member => member;
    public DateOnly StartDate => startDate;
    public DateOnly EndDate => endDate;
    public decimal Fee => fee;
}
```

```
public class Member
{
    private readonly IDictionary<DateOnly, Subscription> _subscriptions
            = new Dictionary<DateOnly, Subscription>();

    public void Subscribe(DateOnly startDate, int months, decimal fee)
    {
        try
        {
            Subscription subscription = new(this, startDate,
startDate.AddMonths(months).AddDays(-1), fee);
            _subscriptions.Add(startDate, subscription);
        }
        catch (ArgumentException)
        {
            throw new VgcException($"Subscription for {startDate} already exists");
        }
    }

    public IDictionary<DateOnly, Subscription> Subscriptions => _subscriptions.AsReadOnly();

    public void AddReservation(BoardGame game)
    {
        _reservedBoardGames.Add(game);
    }

}
```

Many-to-many


Member
*memberNo
name
firstName
reserves
BoardGame
*name
category
gamePublisher

Many-to-many




```
public class Member
{
    private readonly IDictionary<DateOnly, Subscription> _subscriptions
        = new Dictionary<DateOnly, Subscription>();
    private readonly IList<BoardGame> _reservedBoardGames = new List<BoardGame>();
}
```

```
public void ReserveBoardGame(BoardGame boardGame, Member member)
{
    boardGame.AddReservation(member);
    member.AddReservation(boardGame);
}
```

Facade

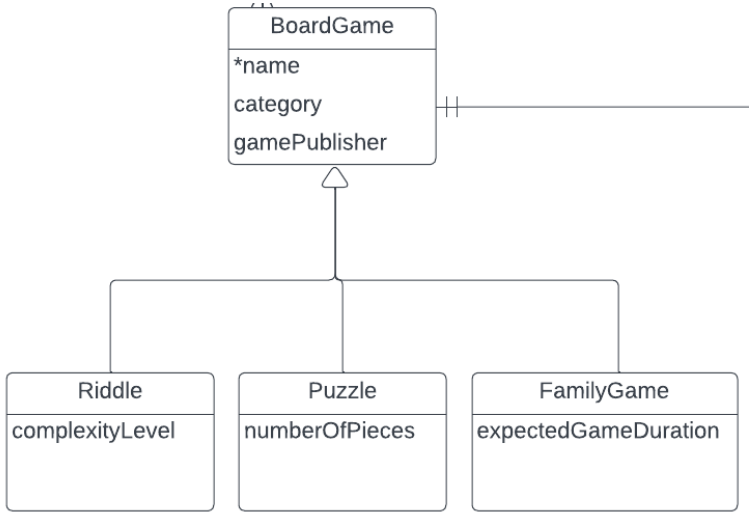
```
public class BoardGame(int boardGameId, string name, Category category, string gamePublisher)
{
    private readonly IList<Member> _reservingMembers = new List<Member>();
}
```

Inheritance

course: Information Modelling

Inheritance

```
public class FamilyGame(int boardGameId, string name, Category category, string
gamePublisher, int expectedGameDuration, int numberOfPlayers)
    : BoardGame(boardGameId, name, category, gamePublisher)
{
    // ...
}
```



Mapping to a class-based object oriented model

course: Information Modelling