agap2IT

TRAINING CENTER

Domain Driven Design
Module 2
Tatical Domain-Driven Design

Domain Driven Design

Module Overview

Entities

Value Objects

Aggregates

Domain Services

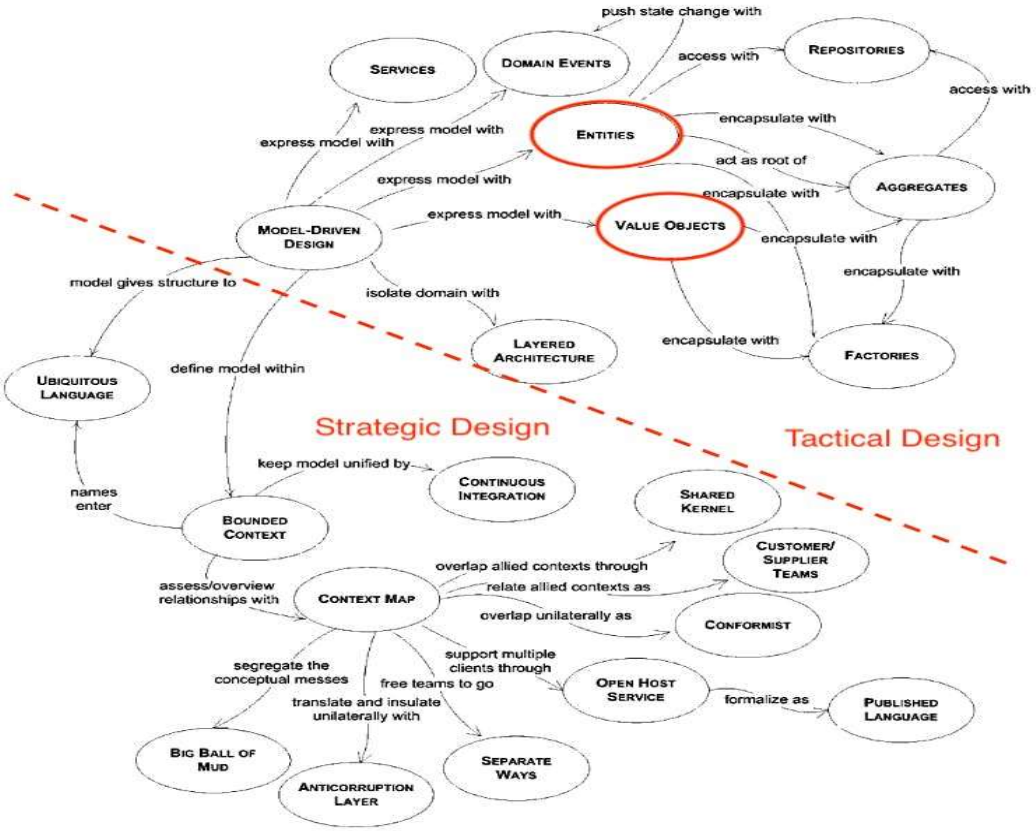
Factories

Domain Driven Design

Introduction

Eric Evans
DDD Author

“DDD can be divided into Strategic and Tactical Design where the Tactical Design is about the building blocks of DDD.”

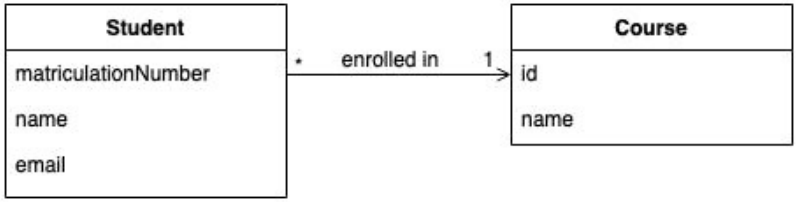


Entities

Eric Evans
DDD Author

“Many objects are not fundamentally defined by their attributes, but rather by a thread of continuity and identity.”

“An object primarily defined by its identity is called an Entity.”



Object diagram

student: Student
matriculationNumber = 1
name = "Wempe"
email = "mail@example.com"

Database representation

STUDENTS		
MATRICULATION_NUMBER	NAME	EMAIL
1	Wempe	mail@eample.com

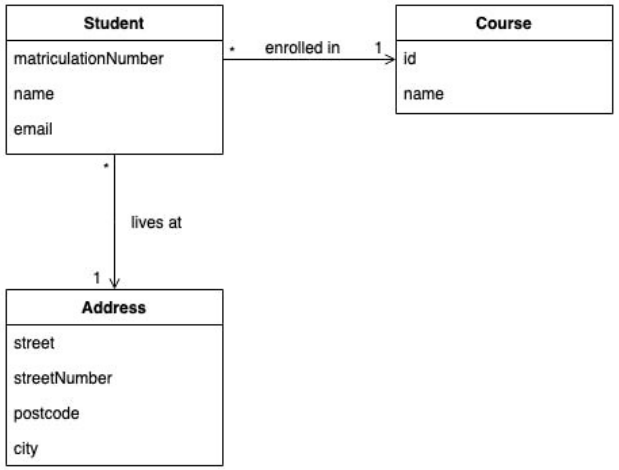
Domain Driven Design

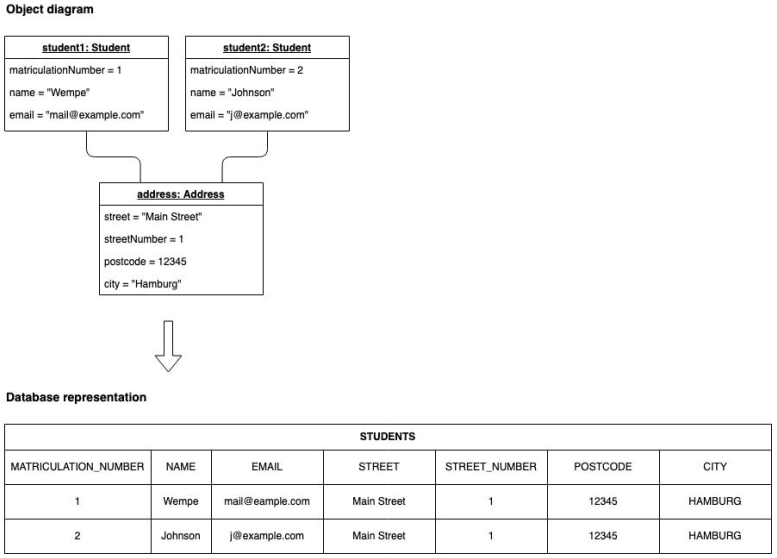
Value Objects

Eric Evans
DDD Author

“Many objects have no conceptual identity. These objects describe some characteristics of a thing.”

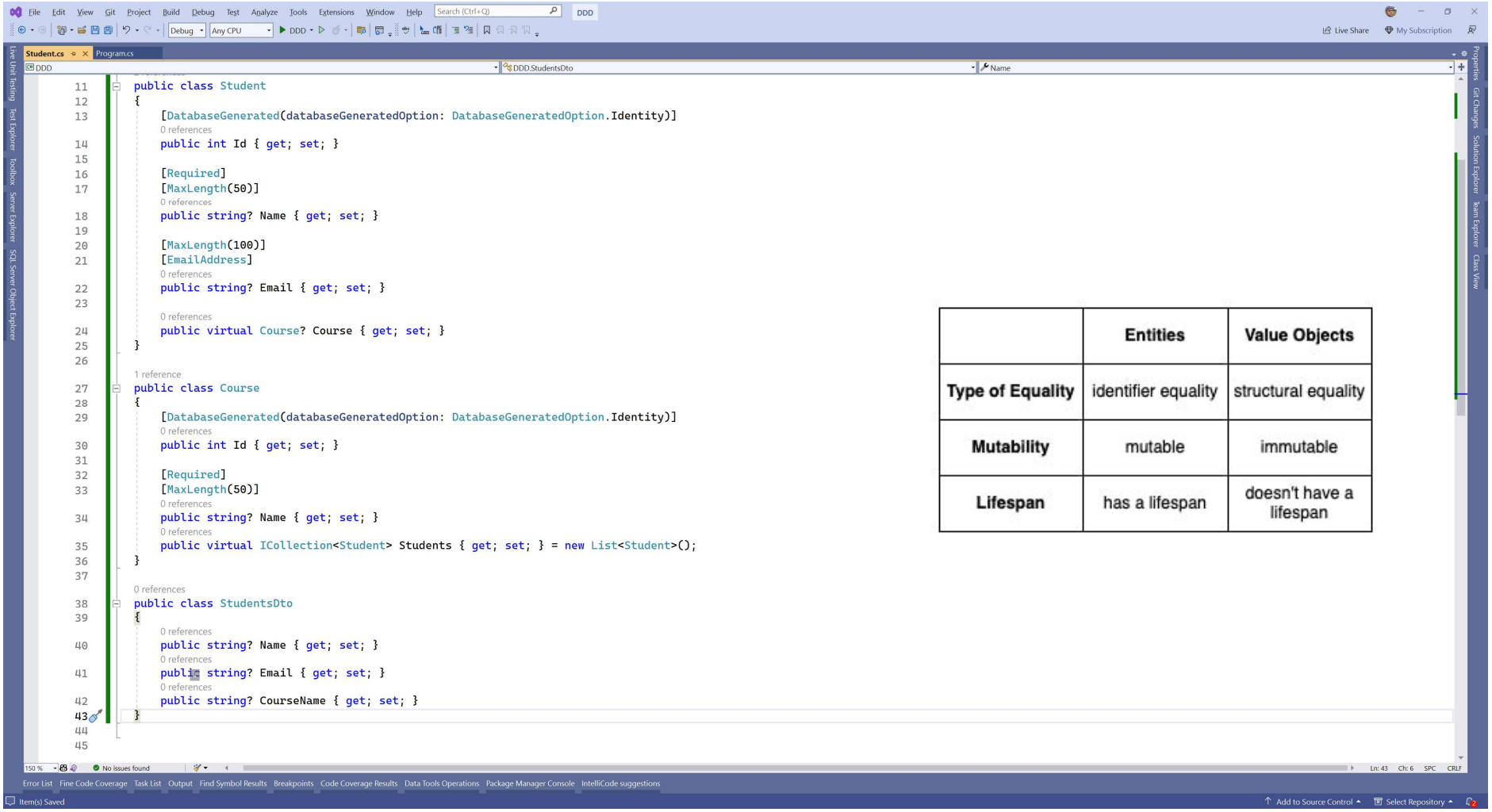
“An object that represents a descriptive aspect of the domain with no conceptual identity is called a Value Object. Value Objects are instantiated to represent elements of the design that we care about only for what they are, not who or which they are..”





STUDENTS						
MATRICULATION_NUMBER	NAME	EMAIL	STREET	STREET_NUMBER	POSTCODE	CITY
1	Wempe	mail@eample.com	Main Street	1	12345	HAMBURG
2	Johnson	j@example.com	Main Street	1	12345	HAMBURG

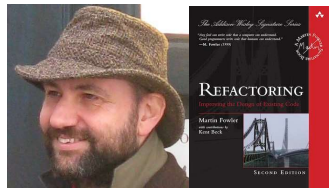
Entities vs Value Objects



```
11  public class Student
12  {
13      [DatabaseGenerated(databaseGeneratedOption: DatabaseGeneratedOption.Identity)]
        0 references
14      public int Id { get; set; }
15
16      [Required]
17      [MaxLength(50)]
        0 references
18      public string? Name { get; set; }
19
20      [MaxLength(100)]
21      [EmailAddress]
        0 references
22      public string? Email { get; set; }
23
        0 references
24      public virtual Course? Course { get; set; }
25  }
26
    1 reference
27  public class Course
28  {
29      [DatabaseGenerated(databaseGeneratedOption: DatabaseGeneratedOption.Identity)]
        0 references
30      public int Id { get; set; }
31
32      [Required]
33      [MaxLength(50)]
        0 references
34      public string? Name { get; set; }
        0 references
35      public virtual ICollection<Student> Students { get; set; } = new List<Student>();
36  }
37
    0 references
38  public class StudentsDto
39  {
        0 references
40      public string? Name { get; set; }
        0 references
41      public string? Email { get; set; }
        0 references
42      public string? CourseName { get; set; }
43  }
44
45
```

	Entities	Value Objects
Type of Equality	identifier equality	structural equality
Mutability	mutable	immutable
Lifespan	has a lifespan	doesn't have a lifespan

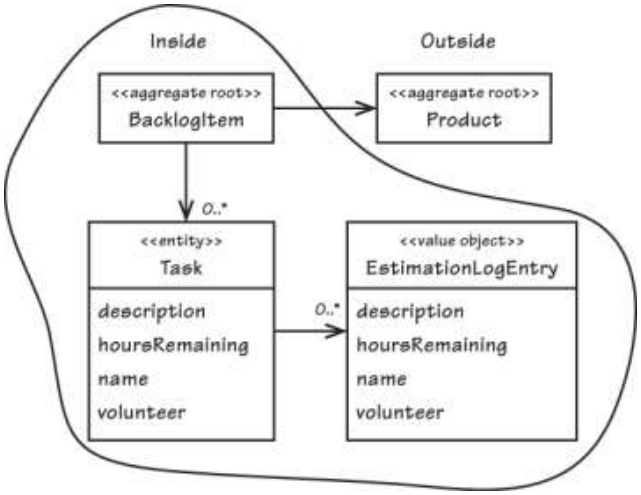
Aggregates



Martin Fowler
Refactoring Author

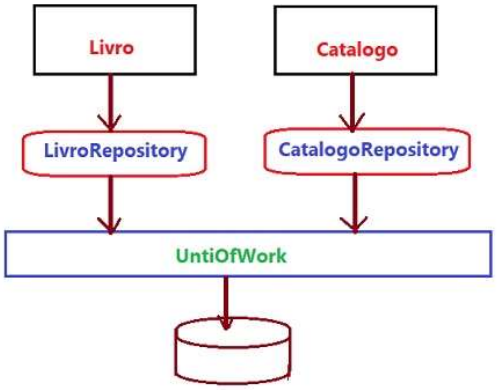
“Aggregate is a pattern in Domain-Driven Design. A DDD aggregate is a cluster of domain objects that can be treated as a single unit. An example may be an order and its line-items, these will be separate objects, but it's useful to treat the order (together with its line items) as a single aggregate.”

“Aggregates are the basic element of transfer of data storage - you request to load or save whole aggregates. Transactions should not cross aggregate boundaries.”



Aggregates

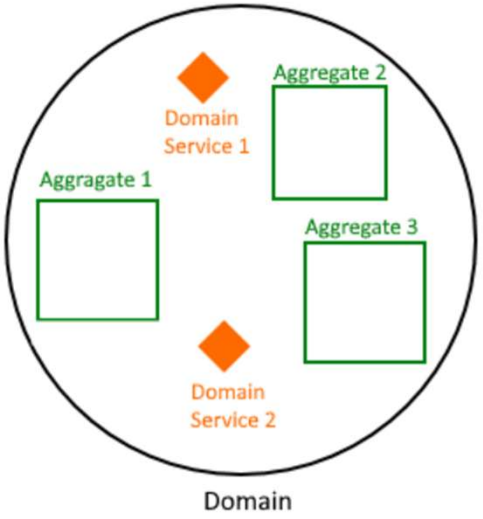
```
1 reference
public class UnitOfWork : IUnitOfWork
{
    private readonly LivrariaDbContext _context;
    1 reference
    public ILivroRepository Livros { get; }
    1 reference
    public ICatalogoRepository Catalogos { get; }

    0 references
    public UnitOfWork(LivrariaDbContext context, ILivroRepository livros, ICatalogoRepository catalogos)
    {
        _context = context ?? throw new ArgumentNullException(nameof(context));
        Livros = livros ?? throw new ArgumentNullException(nameof(livros));
        Catalogos = catalogos ?? throw new ArgumentNullException(nameof(catalogos));
    }

    0 references
    public int Commit()
    {
        return _context.SaveChanges();
    }
}
}
```



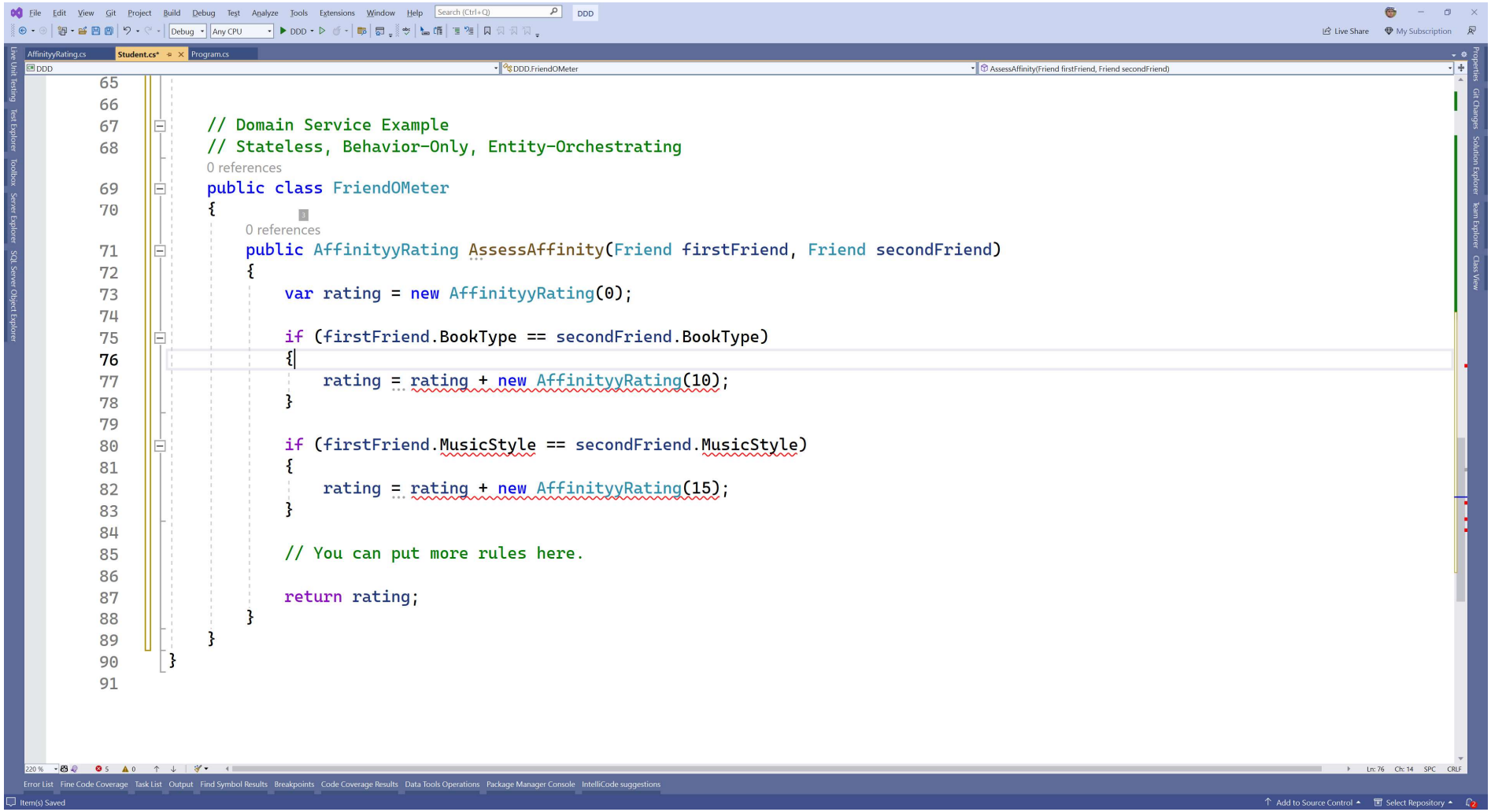
Domain Services

Eric Evans
DDD Author

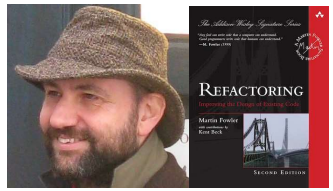
“The Domain Services are Stateless; they reflect behavior and do not have an identity. The Domain Services contain domain logic that does not belong to an Entity or Value Object.”



Domain Services



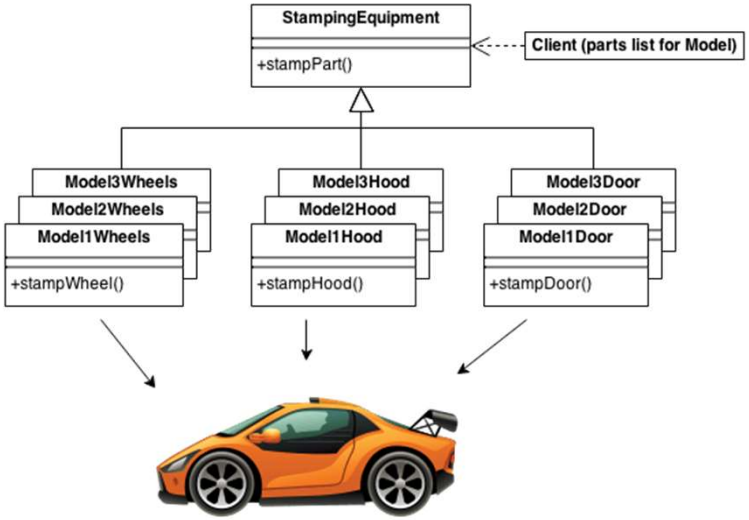
Factories

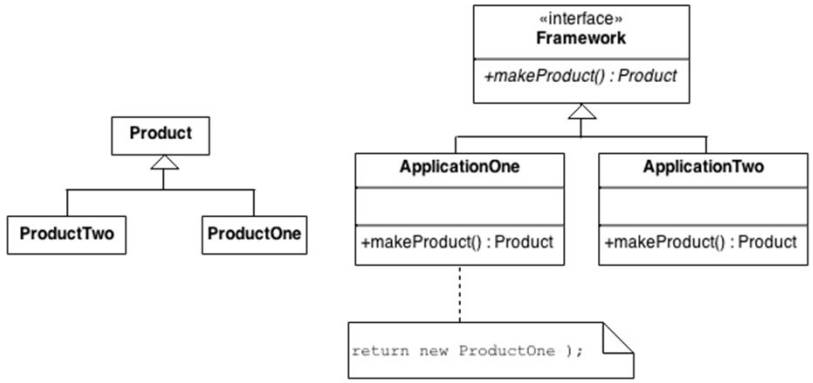


Martin Fowler
Refactoring Author

“A commonly recognized term in object-orientated programming is Factory. A Factory is an object that has the sole responsibility to create other objects.

Factories are not unique to Domain Driven Design, but they do hold an important role within a Domain Driven Design project”





Domain Driven Design

Factories

```
public enum AccountType
{
    Personal = 0,
    Business = 1
}

2 references
public class PersonalAccount : AbstractFactory { }
2 references
public class BusinessAccount : AbstractFactory { }

4 references
public abstract class AbstractFactory
{
    private static readonly PersonalAccount PersonalAccount = new PersonalAccount();
    private static readonly BusinessAccount BusinessAccount = new BusinessAccount();


    0 references
    public static AbstractFactory GetFactory(AccountType accountType)
    {
        AbstractFactory factory;

        switch (accountType)
        {
            case AccountType.Personal:
                factory = PersonalAccount;
                break;
            case AccountType.Business:
                factory = BusinessAccount;
                break;
            default:
                throw new ArgumentOutOfRangeException(nameof(arquitetura), arquitetura, null);
        }

        return factory;
    }
}
```

Who else needs some coffee ?

Domain Driven Design

Technical Training

Let’s do it