ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

КАФЕДРА АВТОМАТИЗАЦИИ ПРЕДПРИЯТИЙ СВЯЗИ

ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

2020

РАЗДЕЛ 1. ВВЕДЕНИЕ В ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ

Предметно-ориентированное проектирование (***Domain Driven Design***, **DDD**) – это набор принципов и схем, направленных на создание оптимальных систем объектов, отражающих предметную область с максимальной степенью полноты и соответствия.

Инструментарий

Visual Studio 2019 Community

SQL Server 2017 Express

Microsoft SQL Server Management Studio 2017

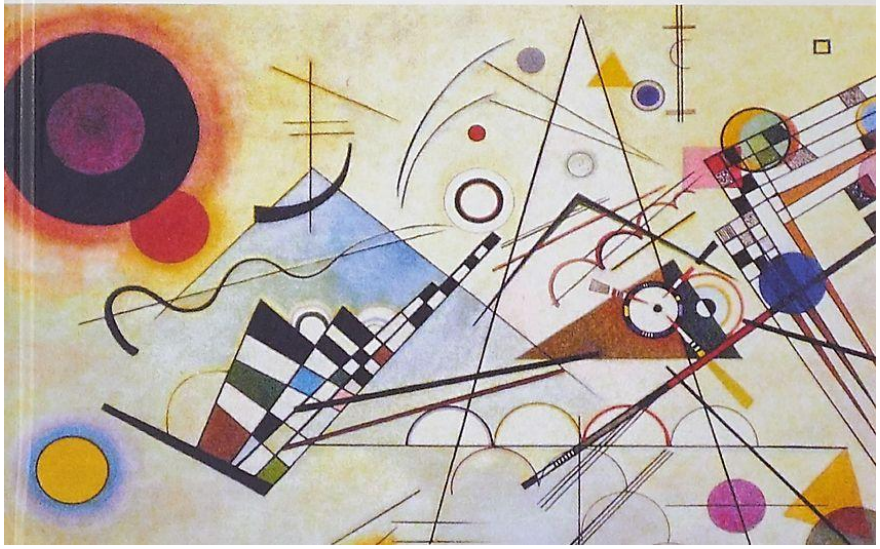
Эрик Эванс

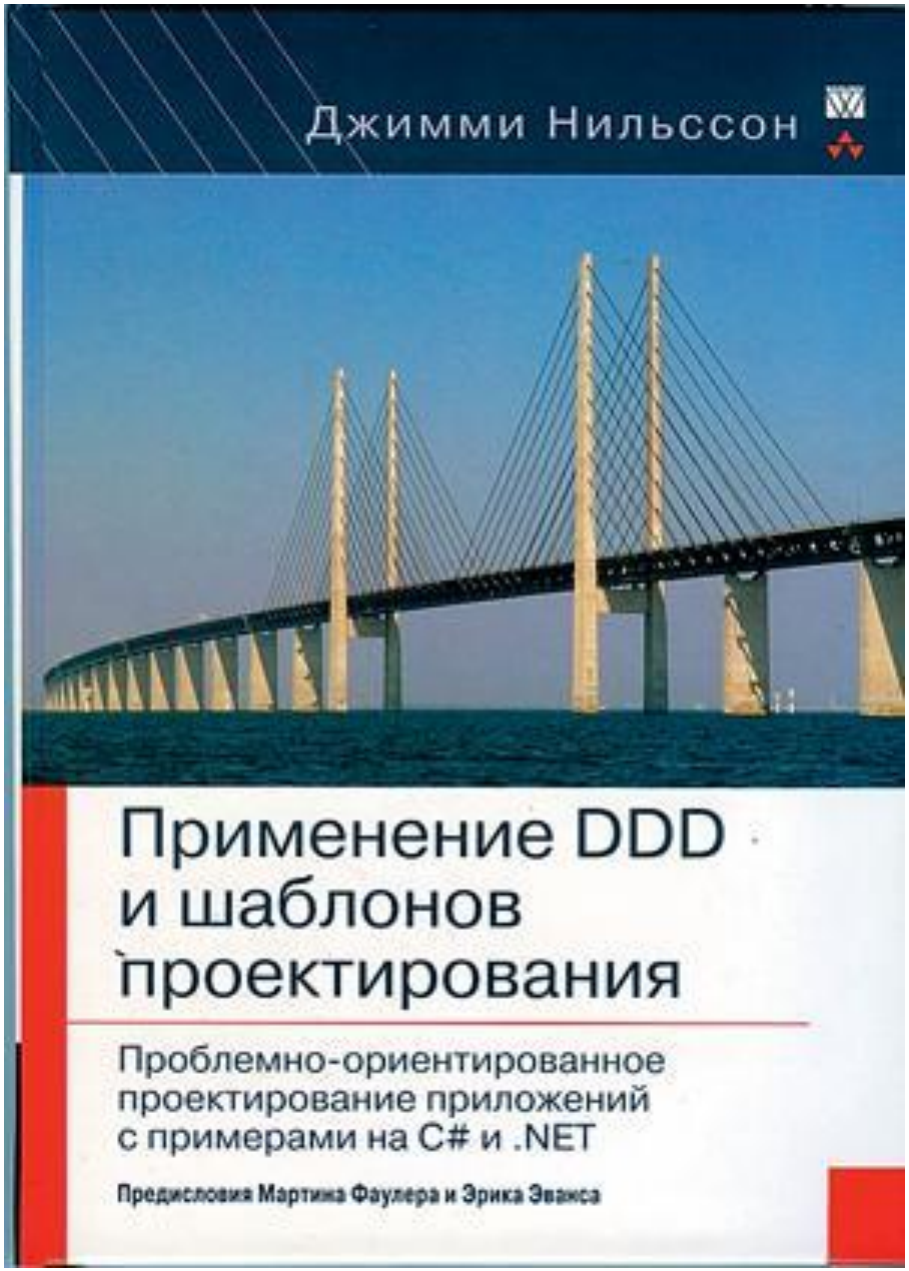
Предисловие
Мартина Фаулера

Domain-Driven
DESIGN

Предметно-ориентированное проектирование

СТРУКТУРИЗАЦИЯ СЛОЖНЫХ ПРОГРАММНЫХ СИСТЕМ




Джимми Нильссон
Применение DDD и шаблонов проектирования
Проблемно-ориентированное проектирование приложений с примерами на C# и .NET
Предисловия Мартина Фаулера и Эрика Эванса

The Addison Wesley Signature Series

Архитектура корпоративных программных приложений

Исправленное издание

Мартин Фаулер
при участии
Дейвида Райса,
Мэттью Фоммела,
Эдварда Хайета,
Роберта Ми
и Рэнди Стаффорда

Объектно-ориентированный анализ и проектирование с примерами приложений

Третье издание

Гради Буч, Роберт А. Максимчук,
Майкл У. Энгл, Бобби Дж. Янг,
Джим Коналлен,
Келли А. Хьюстон

Классы и объекты

Класс точек

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp1
{
    ссылок: 0
    class Point
    {
        public double X;
        public double Y;

        ссылок: 0
        public void MoveBy(double dx, double dy)
        {
            X += dx;
            Y += dy;
        }
    }
}
```

Объекты класса Point

```
using System;

namespace ConsoleApp1
{
    ссылок: 0
    class Program
    {
        ссылок: 0
        static void Main(string[] args)
        {
            Point p1 = new Point();
            p1.X = 5.0;
            p1.Y = 10.0;
            Console.WriteLine("P1.X = {0}, P1.Y = {1}", p1.X, p1.Y);

            Point p2 = new Point();
            p2.X = 15.0;
            p2.Y = 20.0;
            Console.WriteLine("P2.X = {0}, P2.Y = {1}", p2.X, p2.Y);

            p1.MoveBy(50, 100);
            Console.WriteLine("P1.X = {0}, P1.Y = {1}", p1.X, p1.Y);
            Console.WriteLine("P2.X = {0}, P2.Y = {1}", p2.X, p2.Y);
        }
    }
}
```

```
P1.X = 5, P1.Y = 10
P2.X = 15, P2.Y = 20
P1.X = 55, P1.Y = 110
P2.X = 15, P2.Y = 20
```

```
class Point
{
    private double _X;
    ссылок: 7
    public double X
    {
        get { return _X; }
        set { _X = value; }
    }

    private double _Y;
    ссылок: 7
    public double Y
    {
        get { return _Y; }
        set { _Y = value; }
    }

    ссылка: 1
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y += dy;
    }
}
```

Автоматические свойства

```
class Point
{
    ссылок: 7
    public double X { get; set; }
    ссылок: 7
    public double Y { get; set; }

    ссылка: 1
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y += dy;
    }
}
```

Определение свойств с использованием лямбда-выражений

```
class Point
{
    private double _X;
    ссылок: 7
    public double X { get => _X; set => _X = value; }

    private double _Y;
    ссылок: 7
    public double Y { get => _Y; set => _Y = value; }

    ссылка: 1
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y +=dy;
    }
}
```

Конструктор с параметрами

```
class Point
{
    ссылок: 8
    public double X { get; set; }
    ссылок: 8
    public double Y { get; set; }

    ссылок: 2
    public Point (double x, double y)
    {
        X = x;
        Y = y;
    }

    ссылка: 1
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y +=dy;
    }
}
```

Перегрузка конструктора

```
class Point
{
    ссылок: 9
    public double X { get; set; }
    ссылок: 9
    public double Y { get; set; }

    ссылок: 2
    public Point()
    {
        X = 0;
        Y = 0;
    }

    ссылок: 0
    public Point (double x, double y)
    {
        X = x;
        Y = y;
    }

    ссылка: 1
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y +=dy;
    }
}
```

Создание объектов с помощью конструктора с параметрами и без параметров

```
static void Main(string[] args)
{
    Point p1 = new Point();
    p1.X = 5;
    p1.Y = 10;
    Console.WriteLine("P1.X = {0}, P1.Y = {1}", p1.X, p1.Y);

    Point p2 = new Point(15, 20);
    Console.WriteLine("P2.X = {0}, P2.Y = {1}", p2.X, p2.Y);

    p1.MoveBy(50, 100);
    Console.WriteLine("P1.X = {0}, P1.Y = {1}", p1.X, p1.Y);
    Console.WriteLine("P2.X = {0}, P2.Y = {1}", p2.X, p2.Y);
}
```

```
class Program
{
    ссылок: 0
    static void Main(string[] args)
    {
        Point p1 = new Point(5, 10);
        Point p2 = new Point(p1.X, p1.Y);
        Console.WriteLine(p1.Position);

    }
}
```

Копирующий конструктор

```
public Point (Point point)
{
    X = point.X;
    Y = point.Y;
}
```

```
static void Main(string[] args)
{
    Point p1 = new Point(5, 10);
    Point p2 = new Point(p1.X, p1.Y);
    Point p3 = new Point(p2);

    Console.WriteLine(p3.Position);
}
```

Метод клонирования объекта

```
public Point Clone()
{
    return new Point(X, Y);
}
```

```
static void Main(string[] args)
{
    Point p1 = new Point(5, 10);
    Point p2 = new Point(p1.X, p1.Y);
    Point p3 = new Point(p2);
    Point p4 = p3.Clone();

    Console.WriteLine(p1.Position);
    Console.WriteLine(p2.Position);
    Console.WriteLine(p3.Position);
    Console.WriteLine(p4.Position);
}
```

```
X = 5, Y = 10
X = 5, Y = 10
X = 5, Y = 10
X = 5, Y = 10
```

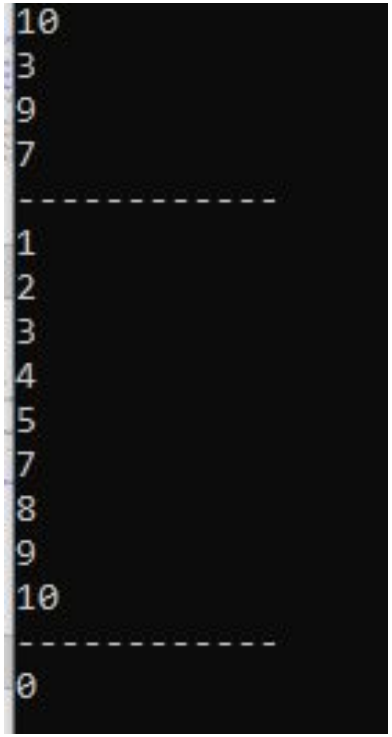
Код класса
Point

```
class Point
{
    ссылок: 8
    public double X { get; set; }
    ссылок: 8
    public double Y { get; set; }

    ссылок: 0
    public Point()
    {
        X = 0;
        Y = 0;
    }

    ссылок: 3
    public Point (double x, double y)
    {
        X = x;
        Y = y;
    }

    ссылка: 1
    public Point (Point point)
    {
        X = point.X;
        Y = point.Y;
    }

    ссылка: 1
    public Point Clone()
    {
        return new Point(X, Y);
    }

    ссылок: 4
    public string Position
    {
        get { return "X = " + X + ", Y = " + Y; }
    }

    ссылок: 0
    public void MoveBy(double dx, double dy)
    {
        X += dx;
        Y +=dy;
    }
}
```

Классы коллекций

```
static void Main(string[] args)
{
    List<int> numbers = new List<int>();
    for(int i = 1; i < 11; i++)
    {
        numbers.Add(i);
    }
    Console.WriteLine(numbers.Count);
    Console.WriteLine(numbers[2]);
    numbers.RemoveAt(5);
    Console.WriteLine(numbers.Count);
    Console.WriteLine(numbers[5]);
    Console.WriteLine("------------");
    foreach(int number in numbers)
    {
        Console.WriteLine(number);
    }
    Console.WriteLine("------------");
    numbers.Clear();
    Console.WriteLine(numbers.Count);
}
```

```
static void Main(string[] args)
{
    List<int> numbers = new List<int>();
    for(int i = 1; i < 11; i++)
    {
        numbers.Add(i);
    }
    Console.WriteLine(numbers.Count);
    Console.WriteLine(numbers[2]);
    numbers.RemoveAt(5);
    Console.WriteLine(numbers.Count);
    Console.WriteLine(numbers[5]);
    Console.WriteLine("------------");
    foreach(int number in numbers)
    {
        Console.WriteLine(number);
    }
    Console.WriteLine("------------");
    numbers.Clear();
    Console.WriteLine(numbers.Count);
}
```



Работа со списком точек

```
static void Main(string[] args)
{
    List<Point> points = new List<Point>();
    for(int i = 0; i < 11; i++)
    {
        points.Add(new Point(i, i));
    }
    foreach(Point point in points)
    {
        point.MoveBy(10, 10);
    }
    foreach(Point point in points)
    {
        Console.WriteLine(point.Position);
    }
}
```

Параметризованные (обобщенные) классы

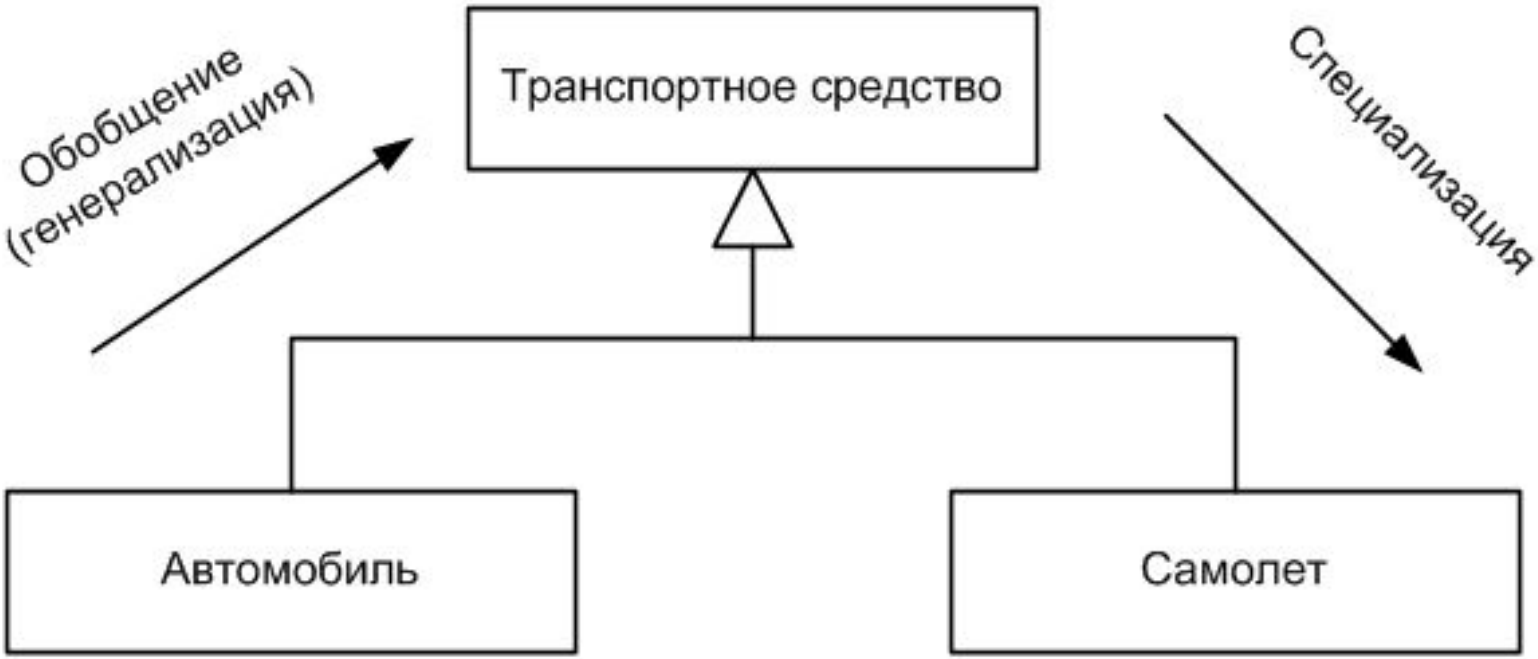
Bingo Cage (пример)

```
public class BingoCage<T>
{
    private Random _Rnd = new Random();

    private List<T> _Cage = new List<T>();
    public List<T> Cage
    {
        get { return _Cage; }
    }

    public void AddItem(T item)
    {
        Cage.Add(item);
    }

    public void AddItems(List<T> items)
    {
        Cage.AddRange(items);
    }
```

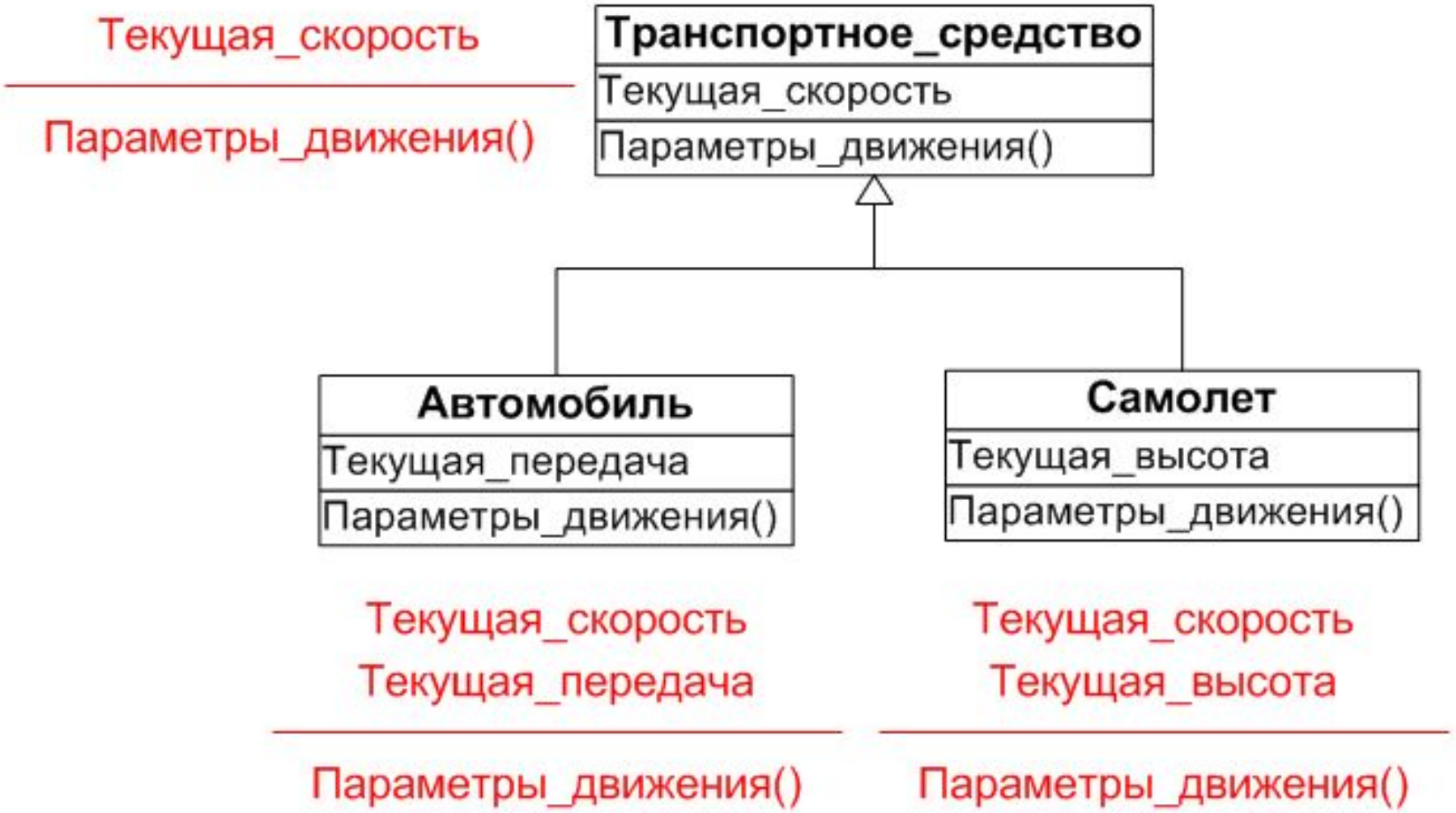
```
public T GetRundomBoll()
{
    int count = Cage.Count;
    int bollIndex;
    if (count > 1)
    {
        bollIndex = _Rnd.Next(0, count);
    }
    else
    {
        bollIndex = 0;
    }
    T boll = Cage[bollIndex];
    Cage.RemoveAt(bollIndex);
    return boll;
}
```

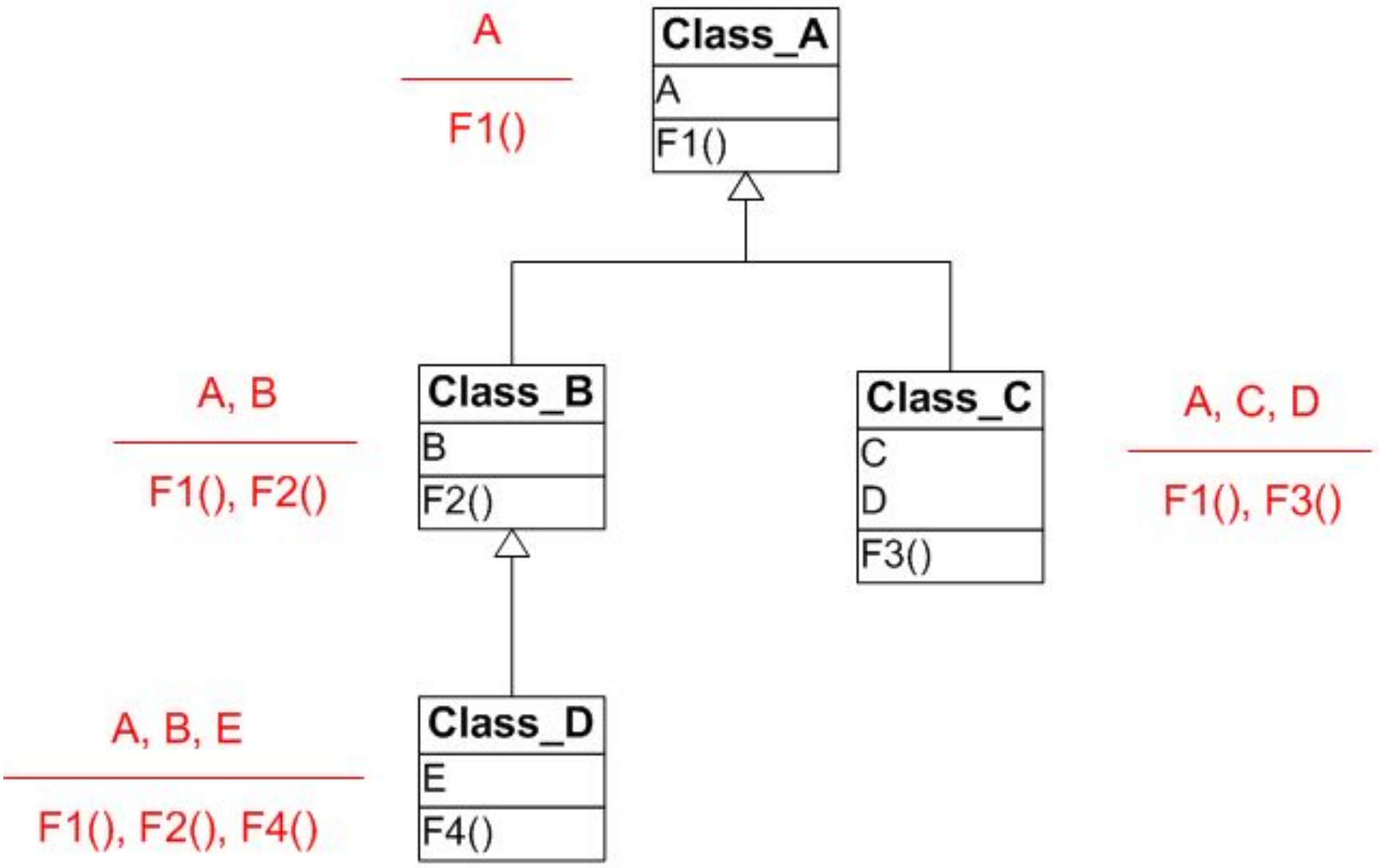
```
    public List<T> GetAllBolls()
    {
        List<T> bolls = new List<T>();
        while(Cage.Count > 0)
        {
            bolls.Add(GetRundomBoll());
        }
        return bolls;
    }

    public List<T> GetBolls(int n)
    {
        List<T> bolls = new List<T>();
        int i = 0;
        while (Cage.Count > 0 && i < n)
        {
            bolls.Add(GetRundomBoll());
            i++;
        }
        return bolls;
    }
}
```

Наследование и полиморфизм


Обобщение
(генерализация)
Транспортное средство
Специализация
Автомобиль
Самолет


Транспортное_средство
Текущая_скорость
Параметры_движения()
Автомобиль
+Текущая_передача
+Параметры_движения()
Самолет
+Текущая_высота
+Параметры_движения()


Текущая_скорость
Параметры_движения()
Транспортное_средство
Текущая_скорость
Параметры_движения()
Автомобиль
Текущая_передача
Параметры_движения()
Самолет
Текущая_высота
Параметры_движения()
Текущая_скорость
Текущая_передача
Параметры_движения()
Текущая_скорость
Текущая_высота
Параметры_движения()


A
F1()
Class_A
A
F1()
A, B
F1(), F2()
Class_B
B
F2()
Class_C
C
D
F3()
A, C, D
F1(), F3()
A, B, E
F1(), F2(), F4()
Class_D
E
F4()

```
public class Транспортное_средство
{
    public int Текущая_скорость { get; set; }

    public virtual string Получить_параметры_движения()
    {
        return "Текущая скорость: " + Текущая_скорость;
    }
}
```

```
public class Автомобиль : Транспортное_средство
{
    public int Текущая_передача { get; set; }

    public override string Получить_параметры_движения()
    {
        return base.Получить_параметры_движения()
            + ", текущая передача: " + Текущая_передача;
    }
}
```

```
public class Самолет : Транспортное_средство
{
    public int Текущая_высота { get; set; }

    public override string Получить_параметры_движения()
    {
        return "Полет на высоте "
            + Текущая_высота
            + " со скоростью " + Текущая_скорость;
    }
}
```

```
List<Транспортное_средство> транспорт = new List<Транспортное_средство>();

транспорт.Add(new Автомобиль { Текущая_скорость = 80, Текущая_передача = 4 });
транспорт.Add(new Автомобиль { Текущая_скорость = 120, Текущая_передача = 4 });
транспорт.Add(new Самолет { Текущая_скорость = 860, Текущая_высота = 10500 });
транспорт.Add(new Автомобиль { Текущая_скорость = 50, Текущая_передача = 3 });
транспорт.Add(new Самолет { Текущая_скорость = 540, Текущая_высота = 4500 });

List<string> параметры_движения = new List<string>();

foreach (Транспортное_средство транспортное_средство in транспорт)
{
    параметры_движения.Add(транспортное_средство.Получить_параметры_движения());
}

GridView1.DataSource = параметры_движения;
Page.DataBind();

Label1.Text = транспорт[0].Текущая_скорость.ToString();
Label2.Text = ((Автомобиль)транспорт[0]).Текущая_передача.ToString();
Label3.Text = ((Самолет)транспорт[2]).Текущая_высота.ToString();
```

Базовые принципы и понятия предметно-ориентированного проектирования

Предметно-ориентированное проектирование (***Domain Driven Design***, **DDD**) – это набор принципов и схем, направленных на создание оптимальных систем объектов, отражающих предметную область с максимальной степенью полноты и соответствия.

Единый язык (ubiquitous language)

В DDD модель создается на едином языке, понятном как программистам, так и экспертам предметной области.

Ubiquitous (англ.) – вездесущий, повсеместный, распространенный, встречающийся повсюду.

Единый язык:

- построен на основе терминов предметной области.
- понятен как программистам и IT-специалистам, так и специалистам предметной области, для которой создается программное обеспечение.

ООП и DDD

ООП

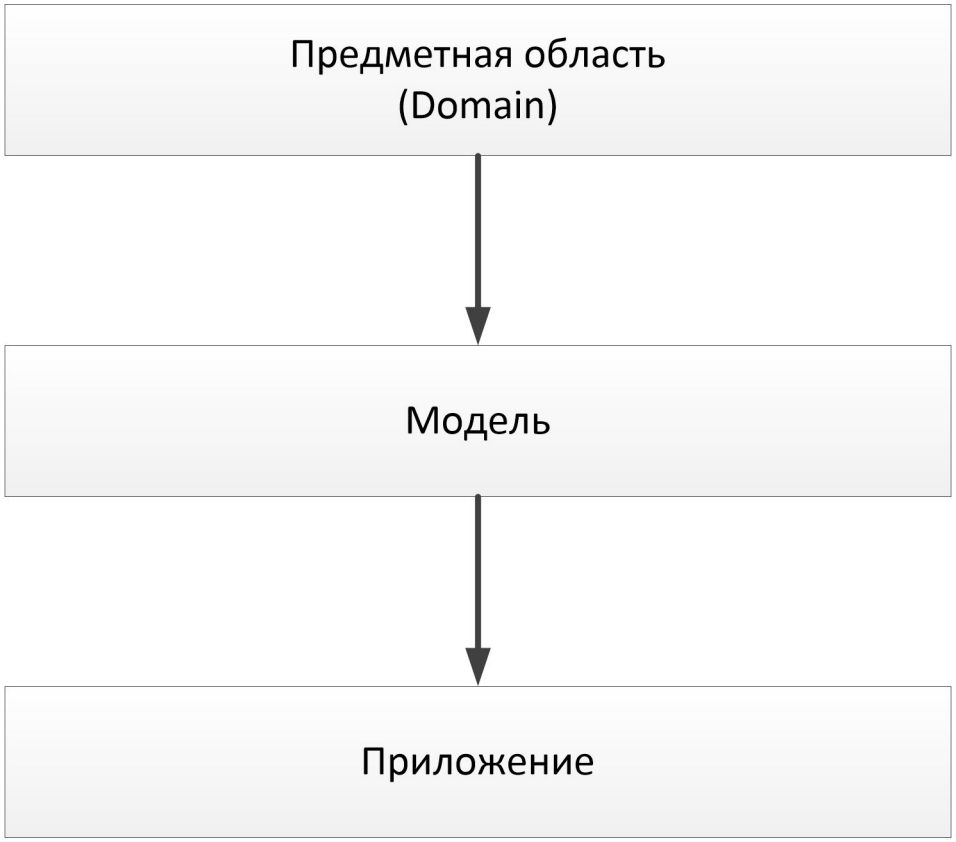
- В объектной модели объекты могут не иметь отношения к предметной области.
- С заказчиком согласуется функционал (use cases, user stories) и интерфейсы (GUI и, при необходимости, взаимодействие с другими системами). Создается обычное ТЗ.

DDD

- Объектная модель предметной области (Domain Model) содержит только элементы, которые соотносятся с объектами предметной области. Инфраструктурные элементы в модели предметной области отсутствуют.
- Модель согласуется с заказчиком. Функционал и интерфейсы следуют из модели.

Управление сложностью в модели предметной области

- Модель верхнего уровня – основные объекты.
- Вспомогательные объекты – на схемах фрагментов, они используются только локально.
- Технические шаблоны в модели не отражаются, достаточно указать их использование.


Предметная область
(Domain)
Модель
Приложение


Предметная область
Модель
Приложение
Предметная область’


Предметная область
Модель
Предметной области‘
Приложение
Предметная область’

Основные понятия DDD

- **Предметная область**, **домен** (англ. ***domain***, домен) – предметная область или область знания, для которой создается программное обеспечение.
- **Модель** (англ. ***model***) – модель предметной области. Описывает отдельные аспекты области и может быть использована для решения проблемы. На ее основе создается программное обеспечение.
- **Единый язык** (англ. ***ubiquitous language***) – используется для описания предметной области (домена) и модели предметной области с единых методологических позиций. Должен быть понятен как разработчикам, так и специалистам в той области, для которой создается программное обеспечение.
- **Ограниченный контекст** (англ. ***bounded context***) – это явная граница, внутри которой существует модель предметной области.

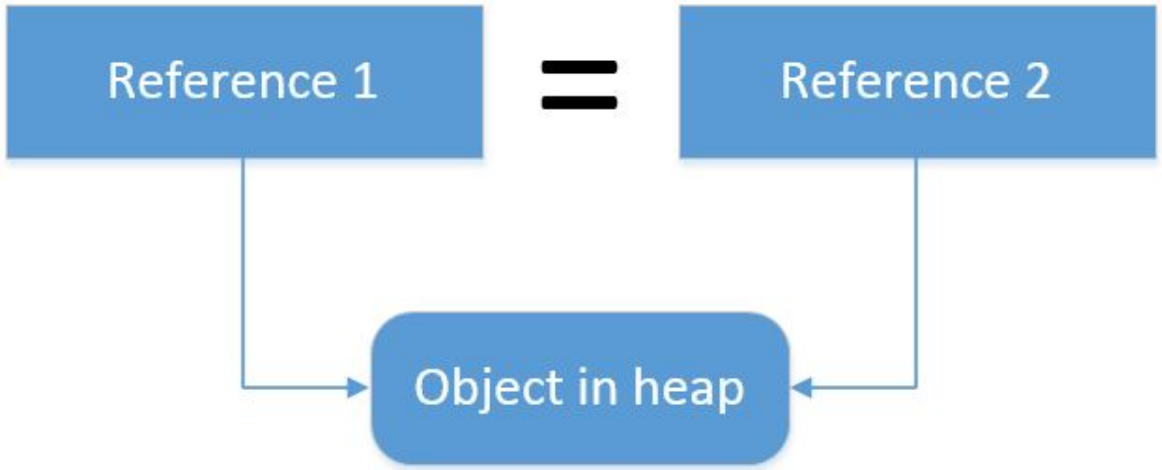
Пример ограниченных контекстов

Контексты (крупномасштабные зоны ответственности) в системе биллинга крупной телекоммуникационной компании:

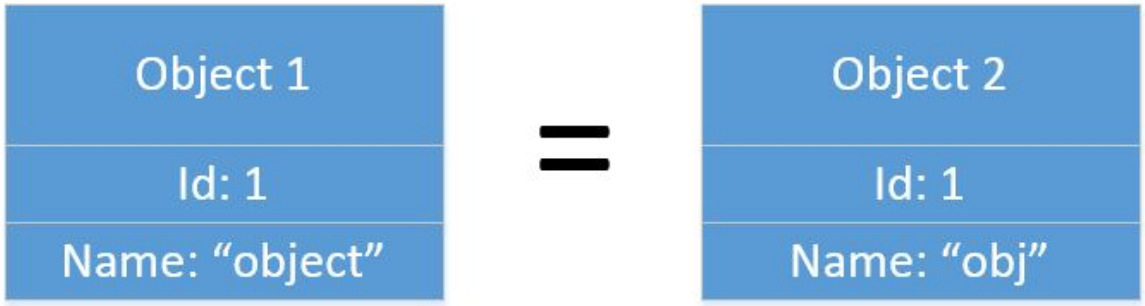
- Клиентская база
- Система безопасности и защиты
- резервное копирование
- Взаимодействие с платежными системами
- Ведение отчётности
- Администрирование
- Система уведомлений

Entity vs Value Object
Сущности и Объекты-значения

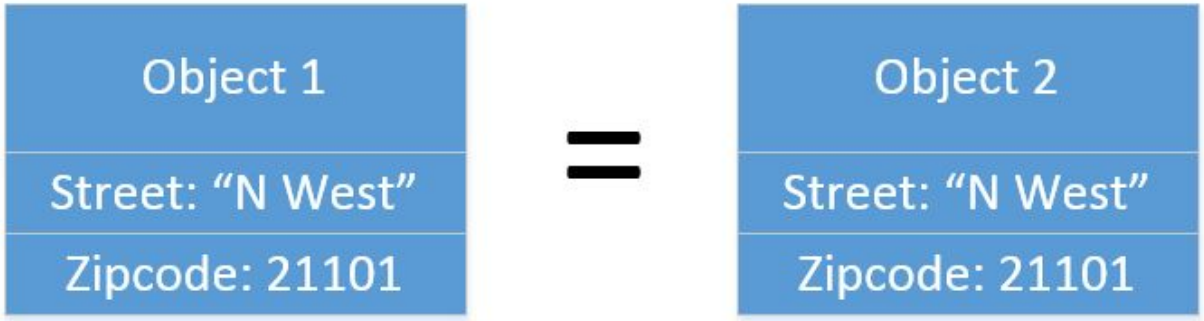
Ссылочная эквивалентность



Identifier equality
Эквивалентность идентификаторов



Структурная эквивалентность



Основное отличие между сущностями и объектами-значения лежит в том, как мы сравниваем их экземпляры друг с другом.

Концепция эквивалентности идентификаторов относится к сущностям, в то время как структурная эквивалентность – к объектам-значениям.

Другими словами, сущности обладают неотъемлемой идентичностью, в то время как объекты-значения – нет.

- Сущности имеют свою собственную, внутренне присущую им идентичность. Объекты-значения — нет.
- Понятие эквивалентности идентификаторов относится к сущностям; понятие структурной эквивалентности — к объектам-значениям; ссылочной эквивалентности — к обоим.
- Сущности имеют историю; у объектов-значений нулевой жизненный цикл.
- Объект-значение всегда должен принадлежать одной или нескольким сущностям, он не может жить собственной жизнью.
- Объекты-значения должны быть неизменяемыми; сущности почти всегда изменяемы.
- Чтобы распознать объект-значение, мысленно замените его на integer.
- Предпочитайте объекты-значения сущностям при моделировании домена.