объектно-ориентированное программирование

принципы объектного проектирования

имутабельность

подход в различных парадигмах

object oriented

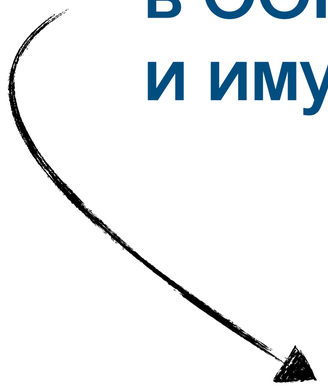
```
class Model
{
    public readonly int Value;

    public Model(int value)
    {
        Value = value;
    }
}
```

≠

functional

```
type Model(value: int) =
    let mutable Value = value
```

свойство данных, не подразумевающее изменения
в ООП, используется в виде сокрытия мутабельных данных
и имутабельность значений не требующих изменения



имутабельность

имутабельность

излишняя мутабельность

```
public class StudentGroup
{
    public long Id { get; set; }

    public string Name { get; set; }

    public List<long> StudentIds { get; set; }

    public void AddStudent(long studentId)
    {
        if (StudentIds.Contains(studentId) is false)
            StudentIds.Add(studentId);
    }
}
```

имутабельность

минимизация мутабельности

```
public class StudentGroup
{
    private readonly HashSet<long> _studentsIds;

    public StudentGroup(long id, string name)
    {
        Id = id;
        Name = name;

        _studentsIds = new HashSet<long>();
    }

    public long Id { get; }

    public string Name { get; set; }

    public IReadOnlyCollection<long> StudentIds => _studentsIds;

    public void AddStudent(long studentId)
    {
        _studentsIds.Add(studentId);
    }
}
```

конвенции

Find/Get

```
public record Post(long Id, string Title, string Content);

public class User
{
    private readonly List<Post> _posts;

    public User(IEnumerable<Post> posts)
    {
        _posts = posts.ToList();
    }
}
```

конвенции

Find/Get

```
public record Post(long Id, string Title, string Content);

public class User
{
    private readonly List<Post> _posts;

    public User(IEnumerable<Post> posts)
    {
        _posts = posts.ToList();
    }

    public Post GetPostById(long postId)
    {
        return _posts.Single(x => x.Id.Equals(postId));
    }

    public Post? FindPostByTitle(string title)
    {
        return _posts.SingleOrDefault(x => x.Title.Equals(title));
    }
}
```

Find/Get

недескриптивный нейминг

```
public Post? FindPost(long postId)
{
    return _posts.Single(x ⇒ x.Id.Equals(postId));
}
public Post? FindPost(string title)
{
    return _posts.SingleOrDefault(x ⇒ x.Title.Equals(title));
}
```

обработка ошибок

исключения

- исключения не отражены в сигнатуре
- поиск конкретного типа исключения и ситуации когда оно кидается приводит к протёкшей абстракции
- неудачное выполнение операции ≠ исключительная ситуация

абстракция, для работы с которой, необходимо иметь знание о деталях её реализации

протёкшая абстракция

обработка ошибок

result types

```
public abstract record AddStudentResult
{
    private AddStudentResult() { }

    public sealed record Success : AddStudentResult;

    public sealed record AlreadyMember : AddStudentResult;

    public sealed record StudentLimitReached(int Limit) : AddStudentResult;
}
```

обработка ошибок

result types

```
public AddStudentResult AddStudent(long studentId)
{
    if (_studentsIds.Count.Equals(MaxStudentCount))
        return new AddStudentResult.StudentLimitReached(MaxStudentCount);

    if (_studentsIds.Add(studentId) is false)
        return new AddStudentResult.AlreadyMember();

    return new AddStudentResult.Success();
}
```

обработка ошибок

result types

```
if (result is AddStudentResult.AlreadyMember)
{
    Console.WriteLine("Student is already member of specified group");
    return;
}

if (result is AddStudentResult.StudentLimitReached err)
{
    var message = $"Cannot add student to specified group, maximum student count of {err.Limit} already reached";
    Console.WriteLine(message);

    return;
}

if (result is not AddStudentResult.Success)
{
    Console.WriteLine("Operation finished unexpectedly");
    return;
}

Console.WriteLine("Student successfully added");
```

domain driven design

value object

```
public class Account
{
    public decimal Balance { get; private set; }

    public void Withdraw(decimal value)
    {
        if (value < 0)
            throw new ArgumentException("Value cannot be negative", nameof(value));

        Balance -= value;
    }
}
```

domain driven design

value object

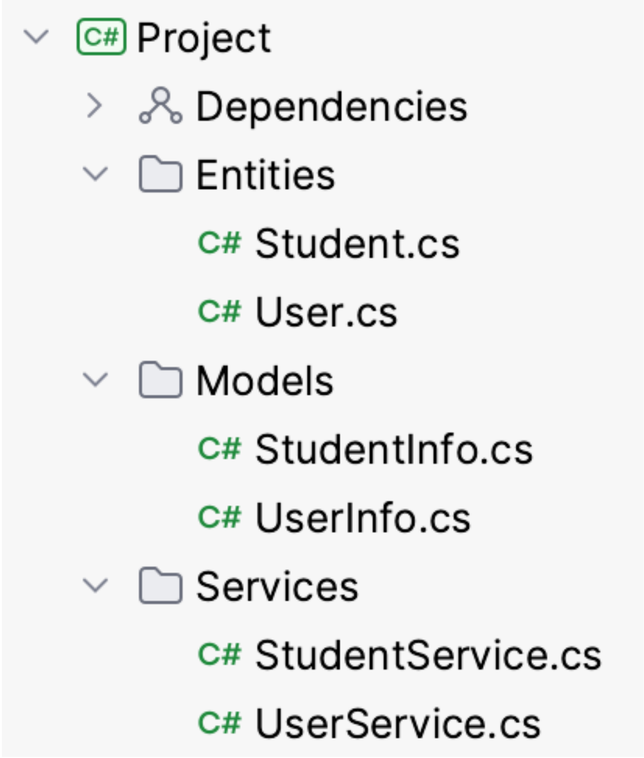
```
public struct Money
{
    public Money(decimal value)
    {
        if (value < 0)
        {
            throw new ArgumentException(
                "Value cannot be negative",
                nameof(value));
        }

        Value = value;
    }

    public decimal Value { get; }

    public static Money operator -(Money left, Money right)
    {
        var value = left.Value - right.Value;
        return new Money(value);
    }
}
```

```
public class Account
{
    public Money Balance { get; private set; }

    public void Withdraw(Money value)
    {
        Balance -= value;
    }
}
```

domain driven design

vertical slices

инфраструктурная



семантическая