Горбачева Маргарита Валерьевна | БПИ-245
Code-review 1

Задачи: проанализировать работы студента_1(только одного студента, ура) и ответить на следующие вопросы:

1. Все ли, по вашему мнению, ключевые сущности предметной области выявлены и присутствуют?
2. Если какие-то из ключевых сущностей отсутствуют, то какие?
3. Насколько автор придерживался условий задания (структура, классы, интерфейсы, связи)? Если автор где-то отклонился от условия / расширил проект, отразилось ли это на качестве программы и как?
4. Адекватен ли уровень абстракции условию задачи?
5. Применены ли в задании принципы ООП и SOLID?
6. Выделите, за счет чего в программе реализуются принципы SOLID? Приведите 2-3 принципа и пример их реализации в проекте автора
7. Выделите, за счет чего в программе реализуются принципы ООП? Приведите 2-3 принципа и пример их реализации в проекте автора
8. Использован ли DI-контейнер в проекте? Как вам кажется, корректно ли он использовался?
9. Что в работе вам понравилось? Какие идеи коллеги вы бы хотели встроить в свои собственные работы в дальнейшем?
10. Что вы можете посоветовать своему коллеге для улучшения качества работы/для развития навыков проектирования?
11. Какую оценку вы бы поставили за работу? Аргументируйте

Code-review первого (и единственного) студента.

1. *Все ли, по вашему мнению, ключевые сущности предметной области выявлены и присутствуют?*

   Хочется начать с того, что очень хорошо осуществлена организация проекта: весь проект разделён на три папки: App, Domain и Tests. Каждая из папок разделена на другие папки.

   Так, в папке App хранится основной файл проекта, так называемый “точка входа приложения”.

   В папке Domain у студента хранятся подпапки Abstractions, Animals, Thigs, Utils. Мегаудобно ориентироваться в чужом проекте, когда у проекта такая прозрачная структура. В свою очередь, подпапки содержат кодовые файлы с уже определенными характеристиками, т.е. В подпапке Abstractions хранятся интерфейсы и абстрактные классы, а в подпапке, например, Animals хранятся

классы конкретных животных - Rabbit, Wolf, Tiger. Также у студента имеется отдельная папка под тесты - Tests, очень приятно.
Что же касается вопроса, какие ключевые сущности предметной области выявлены и присутствуют - это классы конкретных животных(Rabbit, Wolf, Tiger,Monkey, промежуточные классы Predator, Herbo), наследуемые классы инвентаря (сам Thing, Table, Computer), вет.клиника, сам зоопарк, система инвентарного учета.

2. *Если какие-то из ключевых сущностей отсутствуют, то какие?*
   На мой взгляд, все сущности учтены, ничего не потеряно.

3. *Насколько автор придерживался условий задания (структура, классы, интерфейсы, связи)? Если автор где-то отклонился от условия / расширил проект, отразилось ли это на качестве программы и как?*
   Студент учел все условия ТЗ (читать ДЗ), если говорить более подробно, то:
   а) Создано консольное приложение для учета Московского зоопарка - принимать животных на баланс, выводить отчет по количеству животных, потребляемом ими килограммов еды в день, формировать списки тех животных, которые могут быть помещены в контактный зоопарк, выводить наименование и инвентаризационные номера вещей, животных, стоящих на балансе зоопарка.
   б) Структура, классы, интерфейсы - все учтено.
   - Иерархия животных: базовый класс Animal + промежуточные классы - Herbo, Predator - и конкретные животные - Rabbit, Wolf, Tiger, Monkey.
   - Ключевые интерфейсы - IZoo, IVetClinic, IAlive, Inventory.
   - DI-контейнер, см. пункт 1.

Положительные расширения проекта: студент создал файл `Guard.cs`, котрый, как я понимаю, содержит класс для валидации данных, т.е. выбрасывается предупреждение throw new ArgumentException("Value cannot be null or whitespace.", paramName); , если вводится некорректное значение(строка непустая и не состоит из пробелов), или же если речь о инвентаризации, то значение должно быть строго положительным(номера, кол-во еды). Считаю, что данное нововведение положительно отражается на проекте.

4. *Адекватен ли уровень абстракции условию задачи?*
   Полагаю, да. Хорошо реализованы интерфейсы, логика качественная. В папке проекта Abstractions как раз находятся файлы с реализацией интерфейсов предметной логики IAlive для всех живых существ, IInventory для

принадлежности вещи к инвентарю. Абстракции сервисов ветклиники IVetClinic - проверка здоровья животных, абстракция иерархии животных также отражена - Herbo, Predator от Animal, а от них конкретные классы животных. В интерфейсе IZoo наблюдаем подсчет кол-ва животных(ведется их учет), учитывается потребляемое ими кол-во еды в день.

```
public interface IZoo
{
    IReadOnlyCollection<Animal> Animals { get; }
    IReadOnlyCollection<IInventory> Inventory { get; }

    bool TryAcceptAnimal(Animal animal);

    bool RegisterThing(Thing thing);

    int GetTotalDailyFoodKg();

    IEnumerable<Herbo> GetContactZooCandidates();

    int CountAnimals<TAnimal>() where TAnimal : Animal;
}
```

5. *Применены ли в задании принципы ООП и SOLID?*

   Да. Принципы ООП и SOLID применены.В пунктах ниже будет подробное изложение.

6. *Выделите, за счет чего в программе реализуются принципы SOLID? Приведите 2-3 принципа и пример их реализации в проекте автора.*

   1. Принцип единственной ответственности - SRP. Рассмотрим на примере животных.

```
6   public abstract class Animal : IAlive, IInventory
7   {
8       protected Animal(string name, int food, int number)
9       {
10          Guard.AgainstNullOrWhiteSpace(name, nameof(name));
11          Guard.AgainstNonPositive(food, nameof(food));
12          Guard.AgainstNonPositive(number, nameof(number));
13
14          Name = name.Trim();
15          Food = food;
16          Number = number;
17      }
18
19      public string Name { get; }
20      public int Food { get; }
21      public int Number { get; }
```

Так, Animal отвечает только за базовые свойства животных, сюда не нужно добавлять лишние логики. Или же следующий пример на SRP:

```
3   namespace ZooKpoHw1.Domain.Vet;
4
5   public sealed class HealthBasedVetClinic : IVetClinic
6   {
7       public bool CheckHealth(Animal animal) =>
8           animal is not null && animal.Food is >= 1 and <= 100;
9   }
```

Данный класс HealthBasedVetClinic отвечает только за проверку здоровья животных.

2. OCP - принцип открытости/закрытости. Класс ZOO открыт для пополнения.

```
public sealed class Zoo : IZoo
```

```
public bool TryAcceptAnimal(Animal animal)
{
    if (animal is null) throw new ArgumentNullException(nameof(animal));
    if (!_vetClinic.CheckHealth(animal)) return false;
    if (!_usedNumbers.Add(animal.Number)) return false;

    _animals.Add(animal);
    _inventory.Add(animal);
    return true;
}
```

```
public int GetTotalDailyFoodKg() => _animals.Sum(a => a.Food);

public IEnumerable<Herbo> GetContactZooCandidates() =>
    _animals.OfType<Herbo>().Where(h => h.Kindness > 5);

public int CountAnimals<TAnimal>() where TAnimal : Animal =>
    _animals.Count(a => a is TAnimal);
```

3. LSP - дочерние объекты(подклассы) могут заменять классы родителей. Можно привести тот же участок кода, например,

```
public int GetTotalDailyFoodKg() => _animals.Sum(a => a.Food);
```

```
public int CountAnimals<TAnimal>() where TAnimal : Animal =>
    _animals.Count(a => a is TAnimal);
```

```
public bool TryAcceptAnimal(Animal animal)
{
    if (animal is null) throw new ArgumentNullException(nameof(animal));
    if (!_vetClinic.CheckHealth(animal)) return false;
    if (!_usedNumbers.Add(animal.Number)) return false;

    _animals.Add(animal);
    _inventory.Add(animal);
    return true;
}
```

То есть, TryAcceptAnimal принимает любого наследника Animal. Так же и с едой и подсчетом животных, GetTotalDailyFoodKg и CountAnimals работают с любыми наследниками Animal.

```
    Animal animal = kind switch
    {
        1 => new Monkey(name, food, number, ReadBoundedInt("Доброта (0..10): ", 0, 10)),
        2 => new Rabbit(name, food, number, ReadBoundedInt("Доброта (0..10): ", 0, 10)),
        3 => new Tiger(name, food, number),
        4 => new Wolf(name,  food, number),
        _ => throw new InvalidOperationException("Неизвестный вид.")
    };

    bool accepted = zoo.TryAcceptAnimal(animal);
    Console.WriteLine(accepted
        ? $"Принято: {animal}"
        : "Отказано в приёме (ветеринар не одобрил или номер уже используется).");
}
```

Или можно посмотреть на эту часть кода из главного файла Program.cs, где видим, что все конкретные типы подставляются как Animal.

4. ISP - interface segregation principle.

```
namespace ZooKpoHw1.Domain.Abstractions;

public interface IAlive
{
    int Food { get; }
}
```

```
namespace ZooKpoHw1.Domain.Abstractions;

public interface IInventory
{
    int Number { get; }
}
```

Таким образом, видно, что каждый интерфейс отвечает за одну концепцию. IAlive только для живых существ и только свойство питания, а IInventory отвечает только за инвентарные данные.

7. *Выделите, за счет чего в программе реализуются принципы ООП? Приведите 2-3 принципа и пример их реализации в проекте автора.*

1. Инкапсуляция.

Приватные поля в Zoo. Строки private readonly List<Animal> _animals = new(); и private readonly HashSet<int> _usedNumbers = new(); показывают, что внутренняя реализация скрыта.

```
public sealed class Zoo : IZoo
{
    private readonly IVetClinic _vetClinic;
    private readonly List<Animal> _animals = new();
    private readonly List<IInventory> _inventory = new();
    private readonly HashSet<int> _usedNumbers = new();
```

2. Наследование.

Базовый пример с животными, а именно с иерархией наследования. Animal -> Herbo -> Rabbit, например, в проекте можно аналогично увидеть пример с хищниками(Animal -> Predator -> Wolf).

```
public abstract class Herbo : Animal
{
    protected Herbo(string name, int food, int number, int kindness)
        : base(name, food, number)
    {
        Guard.InRange(kindness, 0, 10, nameof(kindness));
        Kindness = kindness;
    }

    public int Kindness { get; }
}
```

```
public sealed class Rabbit : Herbo
{
    public Rabbit(string name, int food, int number, int kindness)
        : base(name, food, number, kindness) { }

    public override string TypeNameRu => "Кролик";
}
```

3. Полиморфизм.

Обработка каждого животного единообразно, каждое животное полиморфно возвращает параметр Food.

```
public int GetTotalDailyFoodKg() => _animals.Sum(a => a.Food);
```

Или вот тут отбор животных с определённым уровнем доброты.

```
public IEnumerable<Herbo> GetContactZooCandidates() =>
    _animals.OfType<Herbo>().Where(h => h.Kindness > 5);
```

8. *Использован ли DI-контейнер в проекте? Как вам кажется, корректно ли он использовался?*

   Да.

```
 8    var services = new ServiceCollection();
 9    services.AddSingleton<IVetClinic, HealthBasedVetClinic>();
10    services.AddSingleton<IZoo, Zoo>();
11    var provider = services.BuildServiceProvider();
12    var zoo = provider.GetRequiredService<IZoo>();
```

   В строках 8-12 главного файла `Guard.cs` видно, что первая строка var services = new ServiceCollection(); создает контейнер зависимостей посредством создания объекта ServiceCollection.
   services.AddSingleton<IVetClinic, HealthBasedVetClinic>(); - тут мы видим регистрацию первой зависимости, которая содержит интерфейс IVetClinic и реализацию HealthBasedVetClinic - класс, который реализует интерфейс и использует механизмы проверки здоровья животных. Далее создается вторая зависимость, а вот тут var provider = services.BuildServiceProvider(); происходит сама сборка контейнера.

```
public sealed class Zoo : IZoo
{
    private readonly IVetClinic _vetClinic;
    private readonly List<Animal> _animals = new();
    private readonly List<IInventory> _inventory = new();
    private readonly HashSet<int> _usedNumbers = new();

    public Zoo(IVetClinic vetClinic)
    {
        _vetClinic = vetClinic ?? throw new ArgumentNullException(nameof(vetClinic));
    }
```

   Здесь видим внедрение зависимостей через конструктор. В строке public Zoo(IVetClinic vetClinic) DI-контейнер передает реализацию.

9. *Что в работе вам понравилось? Какие идеи коллеги вы бы хотели встроить в свои собственные работы в дальнейшем?*

   Я разбила проект на несколько папок, однако не разбивала проект на такие “подробные” папки, как, например, Animal -> Rabbit или Vet -> IVetClinic. Также мне понравилась идея с файлом `Guard.cs`, который содержит класс для валидации данных, т.е. выбрасывается предупреждение throw new ArgumentException("Value cannot be null or whitespace.", paramName); , если

вводится некорректное значение(строка непустая и не состоит из пробелов), или же если речь о инвентаризации, то значение должно быть строго положительным(номера, кол-во еды).

10. *Что вы можете посоветовать своему коллеге для улучшения качества работы/для развития навыков проектирования?*
    Честно говоря, проект мне кажется хорошим. Конечно, быть может, я не вижу серьезных недостатков, но, тем не менее, сказать, что коллеге стоило бы улучшить, не могу. Все хорошо 🙂

11. *Какую оценку вы бы поставили за работу? Аргументируйте*
    10.
    Создано хорошее консольное приложение, соблюдены принципы SOLID и принципы ООП, использовался DI-контейнер, реализованы интерфейсы и классы согласно ТЗ. Даже проведены тесты!! Также мне очень понравилась сама организация папок и подпапок проекта, все интуитивно понятно, несложно, хорошо.