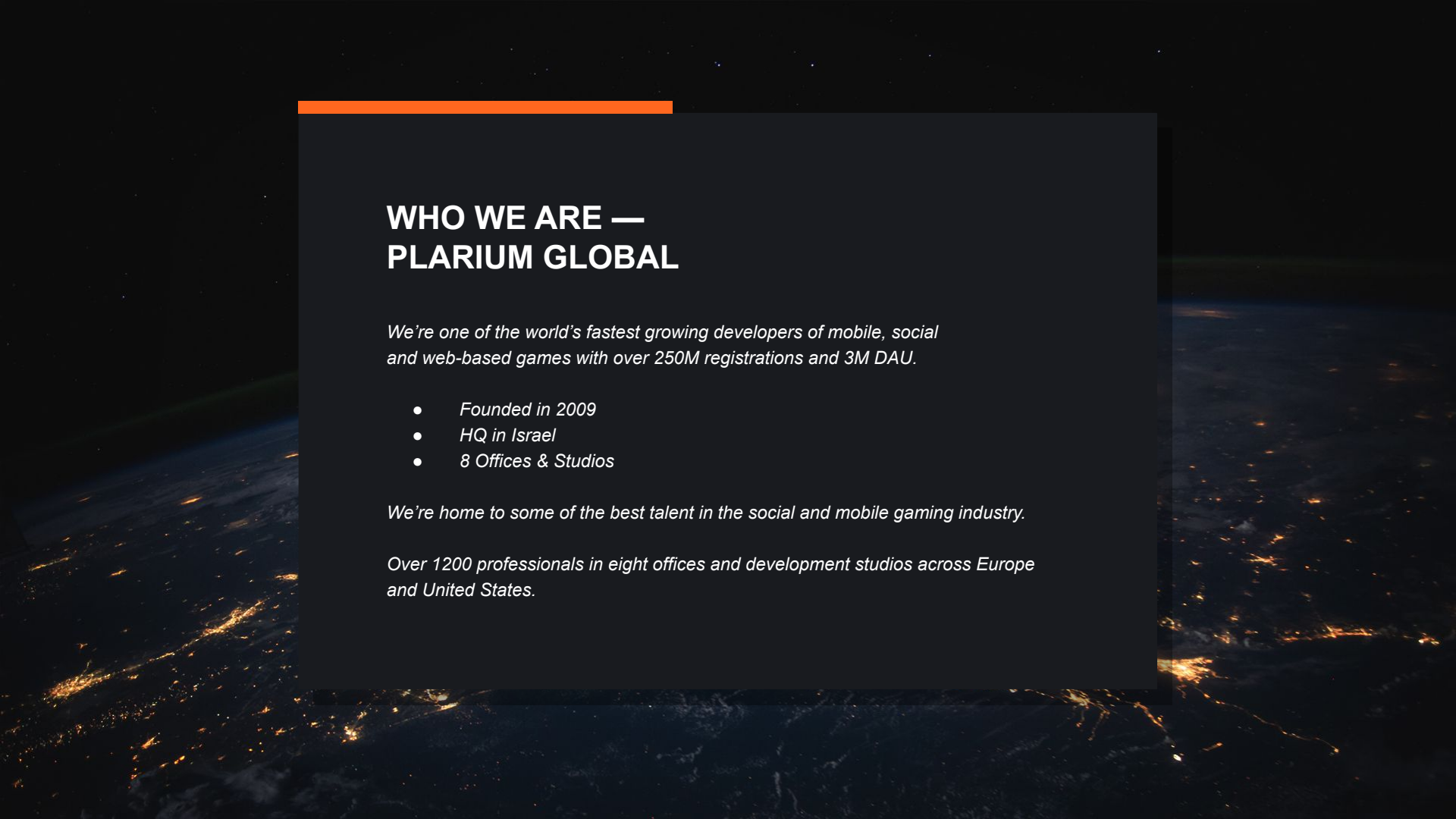

PLARIUM



WHO WE ARE — PLARIUM GLOBAL

We’re one of the world’s fastest growing developers of mobile, social and web-based games with over 250M registrations and 3M DAU.

- *Founded in 2009*
- *HQ in Israel*
- *8 Offices & Studios*

We’re home to some of the best talent in the social and mobile gaming industry.

Over 1200 professionals in eight offices and development studios across Europe and United States.

Проектирование объектов: как открыть бутылку с водой

Igor Stepanov

RPG Department
Client Development Team Lead

Что такое хороший объект?

Что такое объект?

ОПРЕДЕЛЕНИЕ

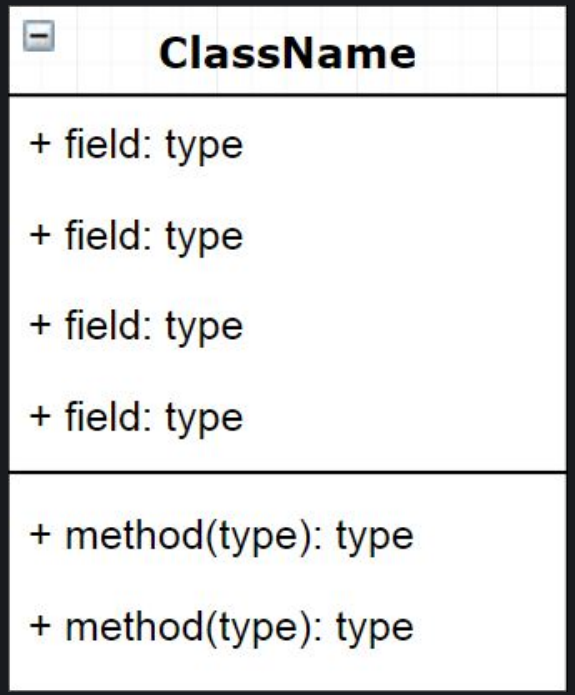
Философская категория, выражающая нечто, на что направлена практическая или познавательная деятельность субъекта (наблюдателя). Это нечто может существовать как в реальной действительности, так и в вымышленном мире; а объектом может быть и сам субъект.

ОПРЕДЕЛЕНИЕ

Философская категория, выражающая нечто, на что направлена практическая или познавательная деятельность субъекта (наблюдателя). Это нечто может существовать как в реальной действительности, так и в вымышленном мире; а объектом может быть и сам субъект.

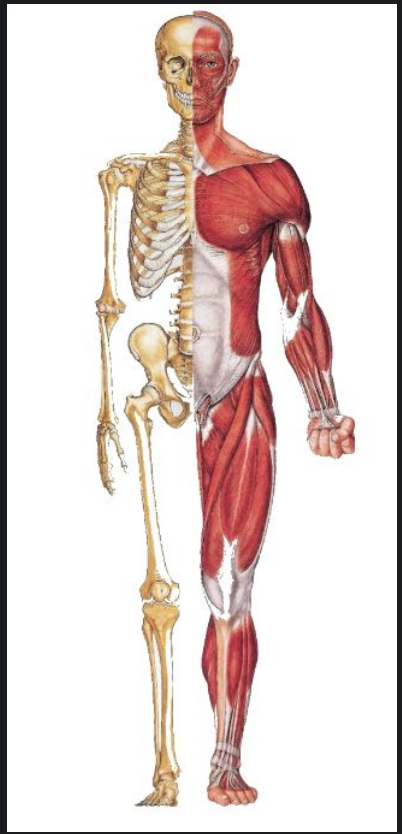
В ПРОГРАММИРОВАНИИ

Некоторая сущность в виртуальном пространстве, обладающая определенным состоянием и поведением, имеет заданные значения свойств (атрибутов) и операций над ними (методов).

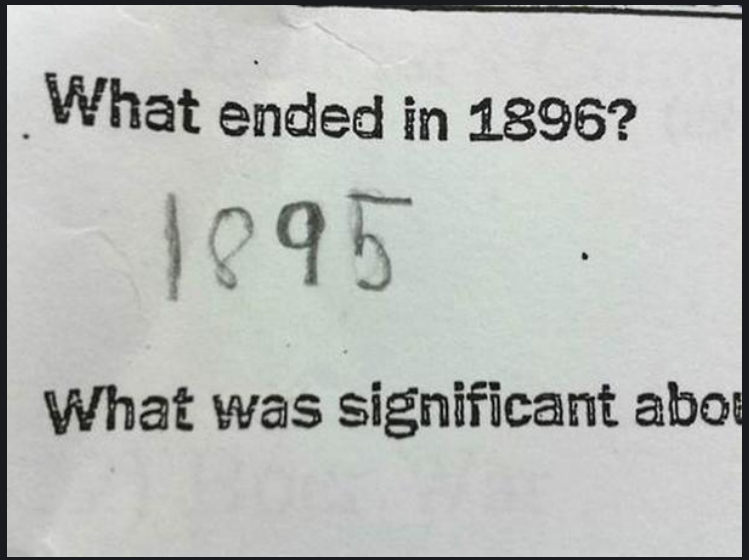


ЧЕЛОВЕК

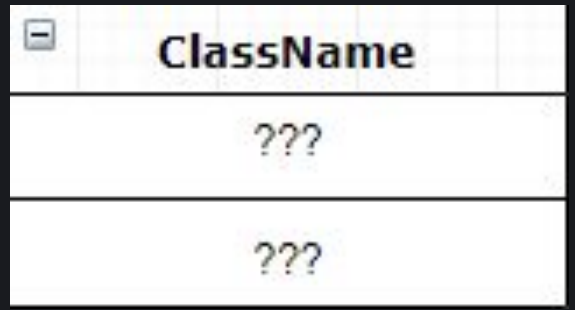
Некоторая сущность в реальном пространстве, обладающая определенным состоянием и поведением; состоит из плоти и костей.



Технически верно...

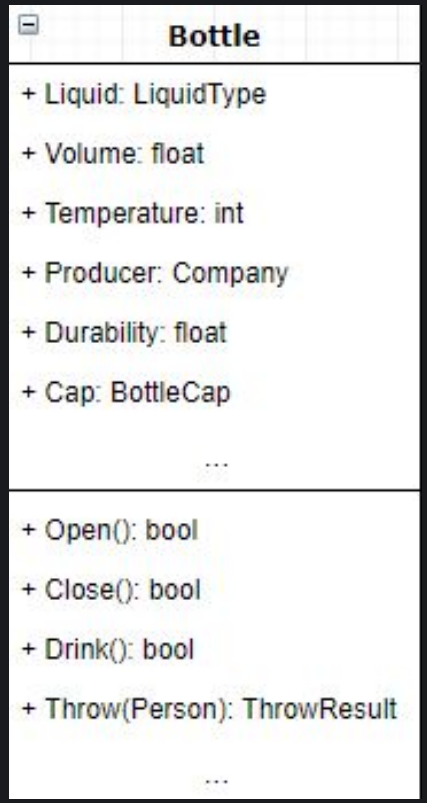


Что такое хороший объект?



Что такое хороший объект?

Что такое хороший объект?



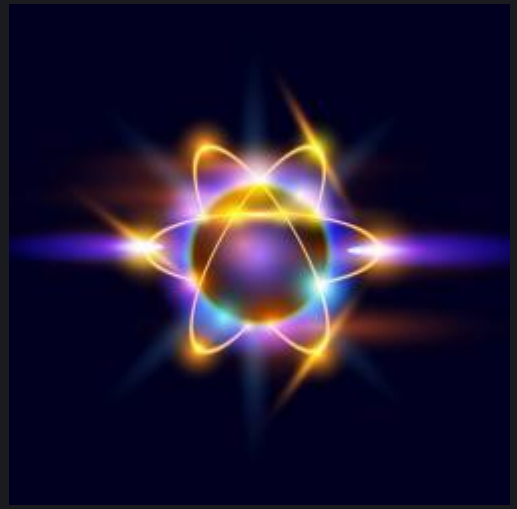
Что объект должен делать в рамках нашего приложения?

Что объект должен делать в рамках нашего приложения?

АНТРОПОМОРФИЗМ

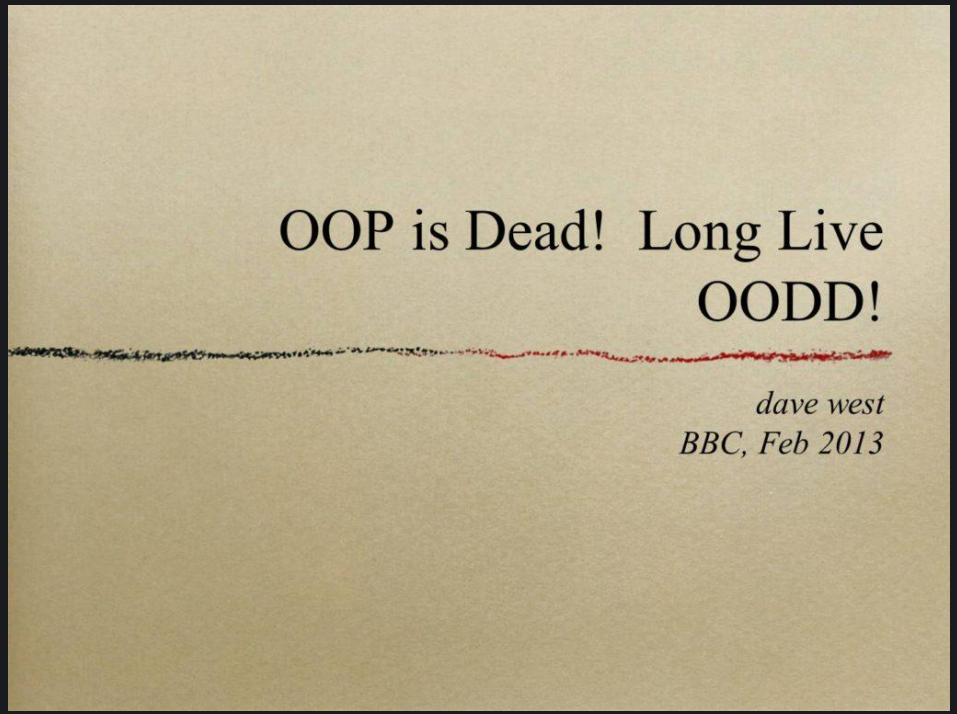
Хороший объект в программировании – это антропоморфная модель объекта предметной области.

Object Thinking by David West, p. 66:
Think of an object as the equivalent of the quanta from which the universe is constructed.

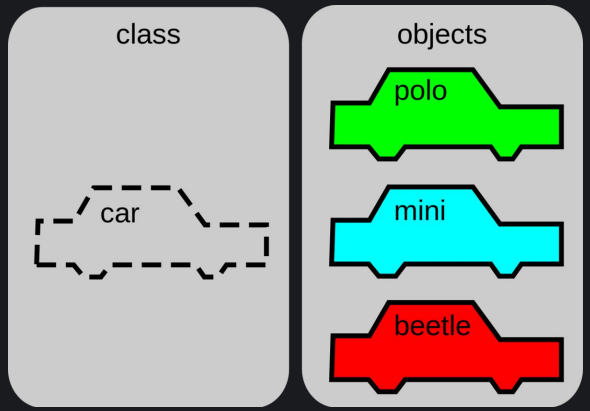


ТРЕБОВАНИЯ

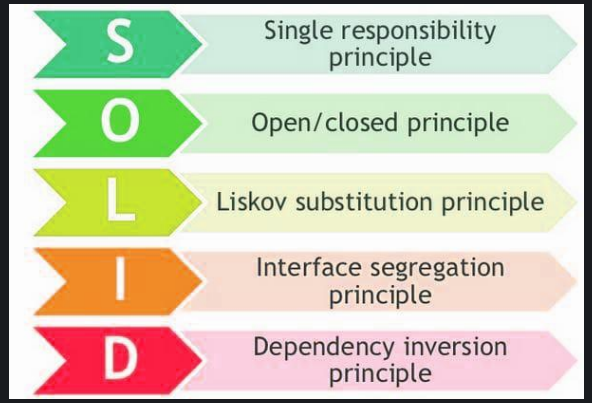
1) Выполняет поставленную задачу
2) Удобен в использовании
3) Устойчив к изменениям
4) Тестируется



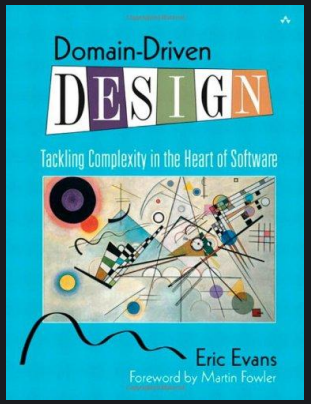
О СКОРОСТИ СТАНОВЛЕНИЯ ИДЕЙ



ООП – 1967 г.

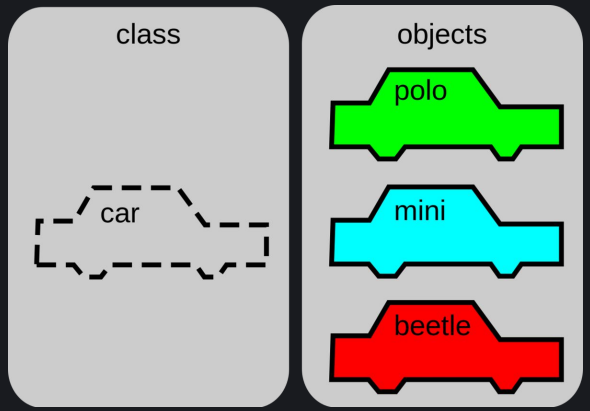


SOLID – 2000 г.

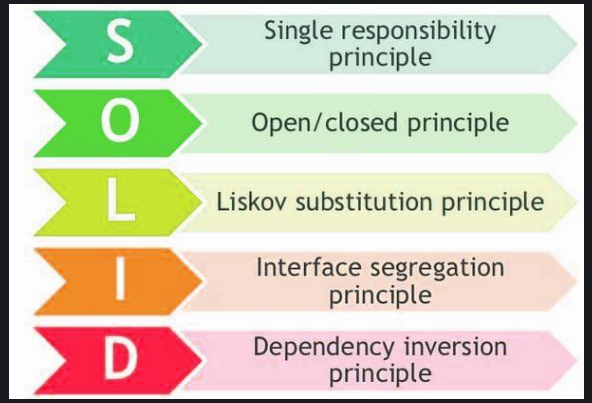


DDD – 2003 г.

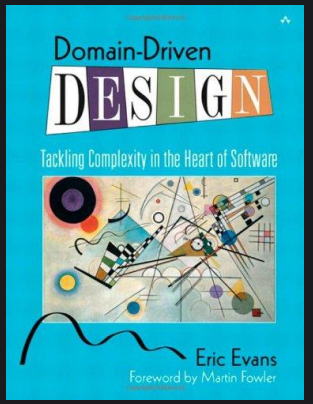
О СКОРОСТИ СТАНОВЛЕНИЯ ИДЕИ?



ООП – 1967 г.



SOLID – 2000 г.



DDD – 2003 г.

ОПРЕДЕЛЕНИЕ

Философская категория, выражающая нечто, на что направлена практическая или познавательная деятельность субъекта (наблюдателя). Это нечто может существовать как в реальной действительности, так и в вымышленном мире; а объектом может быть и сам субъект.

ЭТАПЫ ДЕЯТЕЛЬНОСТИ

1) Вовлечение в деятельность
2) Целеполагание
3) Проектирование действий
4) Осуществление действий
5) Анализ результатов действий и сравнение их с поставленными целями

ПАРАМЕТРЫ ХОРОШЕГО ОБЪЕКТА

1) Существует в предметной области
2) Антропоморфен
3) Обладает лаконичным контрактом данных
4) Уникален и обладает состоянием
5) Не позволяет менять состояние извне
6) Не имеет статики
7) Именуется от сущности, а не от действия

Практические примеры

Практические примеры

```
public interface IPlayer
{
    void Drink(ILiquid liquid);
}
```

```
public interface ILiquid
{
    IEnumerable<IEffect> Effects { get; }
}
```

```
public interface IEffect
{
    void Apply(IPlayer player);
}
```

Использование «данных» о сущности вместо сущности

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;
}
```

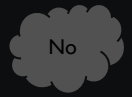

```
public class Bottle
{
    public ILiquid Liquid { get; }

    private Bottle(ILiquid liquid = null) ⇒
        Liquid = liquid;
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;
}
```


```
public class Bottle
{
    public ILiquid Liquid { get; }

    private Bottle(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public static Bottle Empty() ⇒
        new Bottle();
}
```

Использование «данных» о сущности вместо сущности




```
public class BottleData
{
  public ILiquid Liquid { get; }

  public BottleData(ILiquid liquid = null) ⇒
    Liquid = liquid;
}
```


```
public class Bottle
{
  public ILiquid Liquid { get; }

  private Bottle(ILiquid liquid = null) ⇒
    Liquid = liquid;

  ic Bottle Empty() ⇒
  e();
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = Bottle.Empty();
    var bottleData = new BottleData();
  }
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; }

    public BottleData(ILiquid liquid = null) =>
        Liquid = liquid;
}
```


```
public class Bottle
{
    public ILiquid Liquid { get; }

    private Bottle(ILiquid liquid = null) =>
        Liquid = liquid;

    public static Bottle Empty() =>
        new Bottle();
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;
}
```


```
public class Bottle
{
    public ILiquid Liquid { get; private set; }

    private Bottle(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public void FillWith(ILiquid liquid) ⇒
        Liquid = liquid;

    public static Bottle Empty() ⇒
        new Bottle();
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; private set; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public void SetLiquid(ILiquid liquid) ⇒
        Liquid = liquid;
}
```

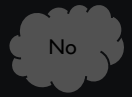

```
public class Bottle
{
    public ILiquid Liquid { get; private set; }

    private Bottle(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public void FillWith(ILiquid liquid) ⇒
        Liquid = liquid;

    public static Bottle Empty() ⇒
        new Bottle();
}
```

Использование «данных» о сущности вместо сущности


```
public class BottleData
{
    public ILiquid Liquid { get; private set; }

    public BottleData(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public void SetLiquid(ILiquid liquid) ⇒
        Liquid = liquid;
}
```


```
public class Bottle
{
    public ILiquid Liquid { get; private set; }

    private Bottle(ILiquid liquid = null) ⇒
        Liquid = liquid;

    public void FillWith(ILiquid liquid) ⇒
        Liquid = liquid;

    public static Bottle Empty() ⇒
        new Bottle();

    public static Bottle With(ILiquid liquid) ⇒
        new Bottle(liquid);
}
```

Использование «данных» о сущности вместо сущности




```
public class BottleData
{
  public ILiquid Liquid { get; private set; }

  public BottleData(ILiquid liquid = null) ⇒
    Liquid = liquid;

  public void SetLiquid(ILiqui
    Liquid = liquid;
}
```


```
public class Bottle
{
  public ILiquid Liquid { get; private set; }

  private Bottle(ILiquid liquid = null) ⇒
    Liquid = liquid;

                                  Liquid liquid) ⇒


                                  mpty() ⇒

                                  With(ILiquid liquid) ⇒
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = Bottle.Empty(); // или Bottle.With(water);
    var bottleData = new BottleData();

    var water = new Water();

    bottle.FillWith(water);
    bottleData.SetLiquid(water);
  }
}
```

Свойства-сеттеры

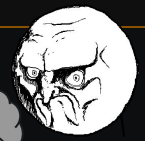
No

```
public class BottleData
{
  public ILiquid Liquid { get; private set; }

  public BottleData(ILiquid liquid = null) ⇒
    Liquid = liquid;

  public void SetLiquid(ILiqui
    Liquid = liquid;
}
```

Yes

```
public class Bottle
{
  public ILiquid Liquid { get; private set; }

  private Bottle(ILiquid liquid = null) ⇒
    Liquid = liquid;
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = Bottle.Empty(); // или Bottle.With(water);
    var bottleData = new BottleData();

    var water = new Water();

    bottle.FillWith(water);
    bottleData.SetLiquid(water); // Может просто засетить свойство?
  }
}
```



No

Объекты, спроектированные «от действия»

Объекты, спроектированные «от действия»


```
public static class BottleOpener
{
  public static ILiquid Open(BottleData bottleData)
  {
    if (bottleData.Liquid == null)
      return null;

    var liquid = bottleData.Liquid;
    bottleData.SetLiquid(null);

    return liquid;
  }
}
```

Объекты, спроектированные «от действия»


```
public static class BottleOpener
{
  public static ILiquid Open(BottleData bottleData)
  {
    if (bottleData.Liquid == null)
      return null;

    var liquid = bottleData.Liquid;
    bottleData.SetLiquid(null);

    return liquid;
  }
}
```


```
public class Bottle
{
  public ILiquid Liquid { get; private set; }

  private Bottle(ILiquid liquid = null) ⇒
    Liquid = liquid;

  public void FillWith(ILiquid liquid) ⇒
    Liquid = liquid;

  public ILiquid Open()
  {
    var liquid = Liquid;
    Liquid = null;
    return liquid;
  }

  public static Bottle Empty() ⇒
    new Bottle();

  public static Bottle With(ILiquid liquid) ⇒
    new Bottle(liquid);
}
```

Объекты, спроектированные «от действия»

Объекты, спроектированные «от действия»


```
internal class Program
{
  public static void Main(string[] args)
  {
    var playerContext = new PlayerContext();
    var waterInfo = new WaterInfo();

    var bottleData = new BottleData(waterInfo);
    var liquidData = BottleOpener.Open(bottleData);

    playerContext.Apply(liquidData);
  }
}
```

Объекты, спроектированные «от действия»


```
internal class ProgramData
{
  public static void Main(string[] args)
  {
    var playerContext = new PlayerContext();
    var waterInfo = new WaterInfo();

    var bottleData = new BottleData(waterInfo);
    var liquidData = BottleOpener.Open(bottleData);

    playerContext.Apply(liquidData);
  }
}
```

Объекты, спроектированные «от действия»


```
internal class ProgramData
{
  public static void Main(string[] args)
  {
    var playerContext = new PlayerContext();
    var waterInfo = new WaterInfo();

    var bottleData = new BottleData(waterInfo);
    var liquidData = BottleOpener.Open(bottleData);

    playerContext.Apply(liquidData);
  }
}
```


```
public static class UserExtensions
{
  public static void Use(this IPlayer self, Bottle bottle)
  {
    var liquid = bottle.Open();
    self.Drink(liquid);
  }
}
```

Объекты, спроектированные «от действия»


```
internal class ProgramData
{
  public static void Main(string[] args)
  {
    var playerContext = new PlayerContext();
    var waterInfo = new WaterInfo();

    var bottleData = new BottleData(waterInfo);
    var liquidData = BottleOpener.Open(bottleData);

    playerContext.Apply(liquidData);
  }
}
```


```
public static class UserExtensions
{
  public static void Use(this IPlayer self, Bottle bottle)
  {
    var liquid = bottle.Open();
    self.Drink(liquid);
  }
}
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var player = new Player();
    var water = new Water();

    var bottle = Bottle.With(water);
    player.Use(bottle);
  }
}
```

Значения перечислений

Значения перечислений


```
public enum SoundTypeId
{
    Default = 0,

    BottleOpened = 1,
    BottleFilled = 2,
    PlayerDamage = 3,
    PlayerHeal = 4
}
```

Значения перечислений


```
public enum SoundTypeId
{
    Empty = 0,

    BottleOpened = 1,
    BottleFilled = 2,
    PlayerDamage = 3,
    PlayerHeal = 4
}
```

Значения перечислений


```
public enum SoundTypeId
{
    Undefined = 0,

    BottleOpened = 1,
    BottleFilled = 2,
    PlayerDamage = 3,
    PlayerHeal = 4
}
```

Значения перечислений


```
public enum SoundTypeId
{
    Undefined = 0,

    BottleOpened = 1,
    BottleFilled = 2,
    PlayerDamage = 3,
    PlayerHeal = 4
}
```

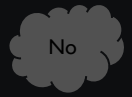

```
public enum SoundTypeId
{
    BottleOpened = 1,
    BottleFilled = 2,
    PlayerDamage = 3,
    PlayerHeal = 4
}
```

Проектирование с учетом контекста

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```

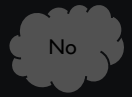
Проектирование с учетом контекста




```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```

```
public class Bottle
{
    public void Open()
    {
        // ...
        SoundManager.PlaySound(SoundTypeId.BottleOpened);
    }
}
```

Проектирование с учетом контекста




```
public static class SoundManager
{
  public static void PlaySound(SoundTypeId typeId)
  {
    // ...
  }
}
```

```
public class Bottle
{
  public void Open()
  {
    // ...
    SoundManager.PlaySound(SoundTypeId.BottleOpened);
  }
}
```

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```

Проектирование с учетом контекста


```
public static class SoundManager
{
  public static void PlaySound(SoundTypeId typeId)
  {
    // ...
  }
}
```


```
public static class Sounds
{
  public static void Play(SoundTypeId typeId)
  {
    // ...
  }
}
```

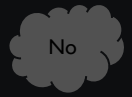
Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```


```
public static class Sounds
{
    public static void Play(SoundTypeId typeId)
    {
        // ...
    }
}
```

```
public static class SoundTypeIdExtensions
{
    public static void Play(this SoundTypeId self) =>
        Sounds.Play(self);
}
```

Проектирование с учетом контекста




```
public static class SoundManager
{
  public static void PlaySound(SoundTypeId typeId)
  {
    // ...
  }
}
```


```
public static class Sounds
{
  public static void Play(SoundTypeId typeId)
  {
```

```
public class Bottle
{
  public void Open()
  {
    // ...
    SoundTypeId.BottleOpened.Play();
  }
}
```

```
SoundTypeIdExtensions

d Play(this SoundTypeId self) ⇒
f);
```

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```


```
public static class Sounds
{
    public static void Play(SoundTypeId typeId)
    {
        // ...
    }
}
```

```
public static class SoundTypeIdExtensions
{
    public static void Play(this SoundTypeId self) =>
        Sounds.Play(self);
}
```

Статические классы


```
public static class SoundManager
{
  public static void PlaySound(SoundTypeId typeId)
  {
    // ...
  }
}
```


```
public class Bottle
{
  private ISounds _sounds;

  public Bottle(ISounds sounds) => // Or any other injection method
    _sounds = sounds;

  public void Open()
  {
    // ...
    Play(SoundTypeId.BottleOpened);
  }

  private void Play(SoundTypeId soundTypeId) =>
    _sounds.Play(soundTypeId);

}
```

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```


```
using static Sandbox.UDev.Correct.Sound.SoundTypeId;

namespace Sandbox.UDev.Correct
{
    public class Bottle
    {
        private ISounds _sounds;

        public Bottle(ISounds sounds) => // Or any other injection method
            _sounds = sounds;

        public void Open()
        {
            // ...
            PlaySound(BottleOpened);
        }

        private void PlaySound(SoundTypeId soundTypeId) =>
            _sounds.Play(soundTypeId);

    }
}
```

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```


```
public class Bottle
{
    private ISounds _sounds;

    public Bottle(ISounds sounds) => // Or any other injection method
        _sounds = sounds;

    public void Open()
    {
        // ...
        _sounds.Play(BottleOpened);
    }
}
```

Проектирование с учетом контекста


```
public static class SoundManager
{
    public static void PlaySound(SoundTypeId typeId)
    {
        // ...
    }
}
```

```
public class Bottle
{
    public void Open()
    {
        // ...
        SoundManager.PlaySound(SoundTypeId.BottleOpened);
    }
}
```


```
public class Bottle
{
    private ISounds _sounds;

    public Bottle(ISounds sounds) => // Or any other injection method
        _sounds = sounds;

    public void Open()
    {
        // ...
        _sounds.Play(BottleOpened);
    }
}
```

Методы обратного вызова

Методы обратного вызова


```
public class BadBottle
{
  private Action _onOpened;

  public BadBottle(Action onOpened) ⇒
    _onOpened = onOpened;

  public void Open()
  {
    // ...
    _onOpened?.Invoke();
  }
}
```

Методы обратного вызова


```
public class BadBottle
{
  private Action _onOpened;

  public BadBottle(Action onOpened) ⇒
    _onOpened = onOpened;

  public void Open()
  {
    // ...
    _onOpened?.Invoke();
  }
}
```


```
public class Bottle
{
  private ISounds _sounds;

  public Bottle(ISounds sounds) ⇒ // Or any other injection method
    _sounds = sounds;

  public void Open()
  {
    // ...
    _sounds.Play(BottleOpened);
  }
}
```

Методы обратного вызова


```
public class BadBottle
{
  private Action _onOpened;

  public BadBottle(Action onOpened) ⇒
    _onOpened = onOpened;

  public void Open()
  {
    // ...
    _onOpened?.Invoke();
  }
}
```


```
public class Bottle
{
  public event Action Opened;

  public void Open()
  {
    // ...
    Opened?.Invoke();
  }
}
```

Подписки на события

Подписки на события


```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = new Bottle();
    bottle.Opened += OnBottleOpened;
  }

  private static void OnBottleOpened()
  {
    // ...
  }
}
```

Подписки на события


```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = new Bottle();
    bottle.Opened +=
  }
}
```

```
() => {}
delegate () {}
Create method BottleOnOpened()
new Action()
null
Magic                    BottleTypeId
Regular                  BottleTypeId
args                     string[]
bottle                   Bottle
Main                     void
Equals                   bool
ReferenceEquals          bool
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>
```

Подписки на события


```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = new Bottle();
    bottle.Opened += OnBottleOpened;
  }

  private static void OnBottleOpened()
  {
    // ...
  }
}
```

Подписки на события


```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = new Bottle();
    bottle.Opened += OnBottleOpened;
  }

  private static void OnBottleOpened()
  {
    // ...
  }
}
```

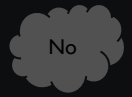

```
internal class Program
{
  public static void Main(string[] args)
  {
    var bottle = new Bottle();
    bottle.Opened += DoSomethingMeaningful;
  }

  private static void DoSomethingMeaningful()
  {
    // ...
  }
}
```

Лишняя бизнес-логика внутри

Лишняя бизнес-логика внутри


```
public class Bottle
{
  private bool _openForbiden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvaliability;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle opening forbidden.");

    // ...
  }

  private void UpdateOpenAvaliability(StageTypeId stageTypeId) ⇒
    _openForbiden = stageTypeId == StageTypeId.ClanBoss;
}
```

Лишняя бизнес-логика внутри




```
public class Bottle
{
  private bool _openForbiden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvaliability;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle ope

    // ...
  }

  private void UpdateOpenAvaliability(StageTypeId sta
    _openForbiden = stageTypeId == StageTypeId.ClanBoss;
}
```

```
public interface IStages
{
  event Action<StageTypeId> Changed;
}
```

Лишняя бизнес-логика внутри


```
public class Bottle
{
  private bool _openForbiden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvaliability;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle opening forbidden.");

    // ...
  }

  private void UpdateOpenAvaliability(StageTypeId stageTypeId) ⇒
    _openForbiden = stageTypeId == StageTypeId.ClanBoss;
}
```

Лишняя бизнес-логика внутри


```
public class Bottle
{
  private bool _openForbiden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvaliability;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle opening forbidden.");

    // ...
  }

  private void UpdateOpenAvaliability(StageTypeId stageTypeId) ⇒
    _openForbiden = stageTypeId == StageTypeId.ClanBoss
                 || stageTypeId == StageTypeId.Dungeon;
}
```

Лишняя бизнес-логика внутри


```
public class Bottle
{
  private bool _openForbiden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvaliability;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle opening forbidden.");

    // ...
  }

  private void UpdateOpenAvaliability(StageTypeId stageTypeId) ⇒
    _openForbiden = stageTypeId == StageTypeId.ClanBoss
                 || stageTypeId == StageTypeId.Dungeon
                 || stageTypeId == StageTypeId.Story;
}
```

Лишняя бизнес-логика внутри


```
public class Bottle
{
  private bool _openForbidden = false;

  public Bottle(IStages stages) ⇒
    stages.Changed += UpdateOpenAvailability;

  private void UpdateOpenAvailability(StageTypeId stageTypeId)
  {
    _openForbidden = stageTypeId == StageTypeId.ClanBoss
                  || stageTypeId == StageTypeId.Dungeon
                  || stageTypeId == StageTypeId.Story;

    if (SomeDebuffActive())
    {
      _openForbidden = false;
    }
    else if (SomeRuleNobodyCaresAbout())
    {
      var fakeOpened = StrangeBudinessRule1() && StrangeBudinessRule2();
      _openForbidden ^= fakeOpened;
    }

  }

  private bool SomeDebuffActive(){ ... }
  private bool SomeRuleNobodyCaresAbout(){ ... }
  private bool StrangeBudinessRule1(){ ... }
  private bool StrangeBudinessRule2(){ ... }
}
```

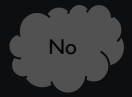

```
public class Bottle
{
  private bool _openForbiden = false;

  public void Open()
  {
    if (_openForbiden)
      throw new InvalidOperationException("Bottle opening forbidden.");

    // ...
  }

  public void ForbidOpening() ⇒
    _openForbiden = true;

  public void AllowOpening() ⇒
    _openForbiden = true;
}
```

Использование паттернов


```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class HoneySyrupDecorator : SyrupDecorator
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```

Использование паттернов




```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class H
{
  protected ov
  {
    new Increa
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```

```
public static void Main(string[] args)
{
  var liquid = new HoneySyrupDecorator(new MapleSyrupDecorator(new Water()));
}
```

Использование паттернов


```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

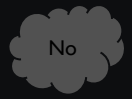
```
public class HoneySyrupDecorator : SyrupDecorator
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```


```
public abstract class Syrup : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private ILiquid _other;

  public ILiquid MixedWith(ILiquid other)
  {
    _other = other;
    return this;
  }
}
```

```
public class HoneySyrup : Syrup
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };
}
```

```
public static class LiquidExtensions
{
  public static ILiquid With(this ILiquid self, Syrup syrup) ⇒
    syrup.MixedWith(self);
}
```

Использование паттернов




```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class HoneySyrupDecorator : Syrup
{
  protected override IEnumerable<IEffect
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid othe
  {
  }
}
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var liquid = new Water()
      .With(new HoneySyrup())
      .With(new MapleSyrup());
  }
}
```

```
public abstract class Syrup : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  quid other)
```

```
 Syrup

merable<IEffect> AdditionalEffects ⇒ new[]

)
```

```
public static class LiquidExtensions
{
  public static ILiquid With(this ILiquid self, Syrup syrup) ⇒
    syrup.MixedWith(self);
}
```

Использование паттернов


```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class HoneySyrupDecorator : SyrupDecorator
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```


```
public abstract class Syrup : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private ILiquid _other;

  public ILiquid MixedWith(ILiquid other)
  {
    _other = other;
    return this;
  }
}
```

```
public class HoneySyrup : Syrup
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };
}
```

```
public static class LiquidExtensions
{
  public static ILiquid With(this ILiquid self, Syrup syrup) ⇒
    syrup.MixedWith(self);
}
```

Использование паттернов


```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class HoneySyrupDecorator : SyrupDecorator
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```


```
public abstract class Syrup : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private ILiquid _other;

  public ILiquid MixedWith(ILiquid other)
  {
    _other = other;
    return this;
  }
}
```

```
public class HoneySyrup : Syrup
{
  protected override IEnumerable<IEffect> AdditionalEffects ⇒ new[]
  {
    new IncreasedStamina()
  };
}
```

```
public static class LiquidExtensions
{
  public static ILiquid With<TSyrup>(this ILiquid self)
    where TSyrup : Syrup, new() ⇒
    new TSyrup().MixedWith(self);
}
```

Использование паттернов

No

```
public abstract class SyrupDecorator : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private readonly ILiquid _other;

  protected SyrupDecorator(ILiquid other) ⇒
    _other = other;
}
```

```
public class HoneySyrupDecorator : SyrupDecorato
{
  protected override IEnumerable<IEffect> Additi
  {
    new IncreasedStamina()
  };

  public HoneySyrupDecorator(ILiquid other) : base(other)
  {
  }
}
```

```
internal class Program
{
  public static void Main(string[] args)
  {
    var liquid = new Water()
      .With<HoneySyrup>()
      .With<MapleSyrup>();
  }
}
```

Yes

```
public abstract class Syrup : ILiquid
{
  public IEnumerable<IEffect> Effects ⇒ _other.Effects.Union(AdditionalEffects);

  protected abstract IEnumerable<IEffect> AdditionalEffects { get; }

  private ILiquid _other;

  xedWith(ILiquid other)
```

```
eySyrup : Syrup

ride IEnumerable<IEffect> AdditionalEffects ⇒ new[]

dStamina()
  };
}
```

```
public static class LiquidExtensions
{
  public static ILiquid With<TSyrup>(this ILiquid self)
    where TSyrup : Syrup, new() ⇒
    new TSyrup().MixedWith(self);
}
```

Использование паттернов


```
public static void Main(string[] args)
{
    var liquid = new HoneySyrupDecorator(new MapleSyrupDecorator(new Water()));
}
```


```
internal class Program
{
    public static void Main(string[] args)
    {
        var liquid = new Water()
            .With<HoneySyrup>()
            .With<MapleSyrup>();
    }
}
```

Декомпозиция на объекты


```
public class Bottle
{
  public Dictionary<BottleRarity, int> ExperienceByRarity = new Dictionary<BottleRarity, int>
  {
    [BottleRarity.Rare] = 10,
    [BottleRarity.Sacred] = 20,
    [BottleRarity.Legendary] = 30,
  };
  public Dictionary<BottleSize, int> ExperienceBySize  = new Dictionary<BottleSize, int>
  {
    [BottleSize.Small] = 10,
    [BottleSize.Medium] = 20,
    [BottleSize.Big] = 30,
  };
}
```


```
public class Bottle
{
  public BottleType Type{ ... }
}
```

Декомпозиция на объекты


```
public class Bottle
{
  public Dictionary<BottleRarity, int> ExperienceByRarity = new Dictionary<BottleRarity, int>
  {
    [BottleRarity.Rare] = 10,
    [BottleRarity.Sacred] = 20,
    [BottleRarity.Legendary] = 30,
  };
  public Dictionary<BottleSize, int> ExperienceBySize  = new Dictionary<BottleSize, int>
  {
    [BottleSize.Small] = 10,
    [BottleSize.Medium] = 20,
    [BottleSize.Big] = 30,
  };
}
```


```
public class Bottle
{
  public BottleType Type{ ... }
  public Durability Durability{ ... }
}
```

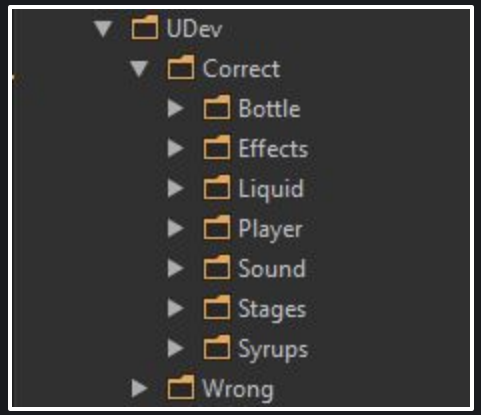
Декомпозиция на объекты

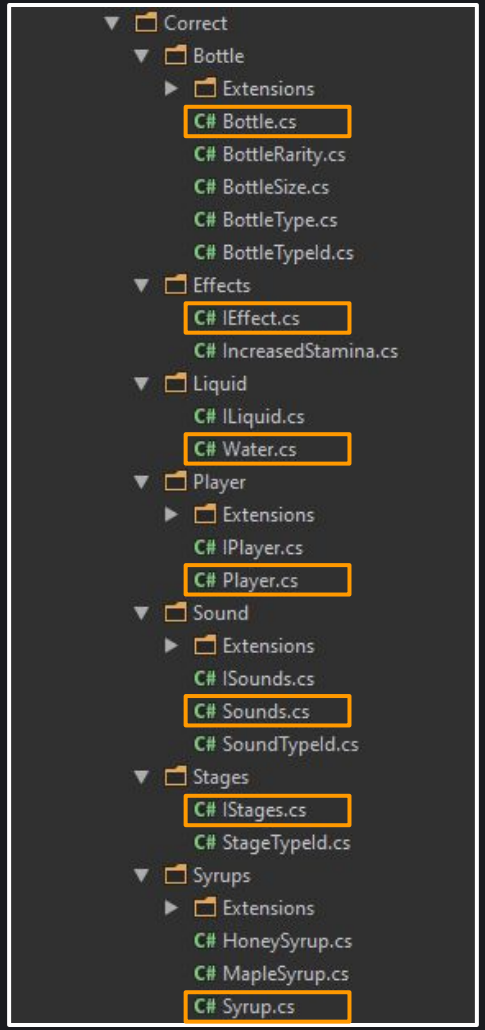

```
public class Bottle
{
  public Dictionary<BottleRarity, int> ExperienceByRarity = new Dictionary<BottleRarity, int>
  {
    [BottleRarity.Rare] = 10,
    [BottleRarity.Sacred] = 20,
    [BottleRarity.Legendary] = 30,
  };
  public Dictionary<BottleSize, int> ExperienceBySize  = new Dictionary<BottleSize, int>
  {
    [BottleSize.Small] = 10,
    [BottleSize.Medium] = 20,
    [BottleSize.Big] = 30,
  };
}
```

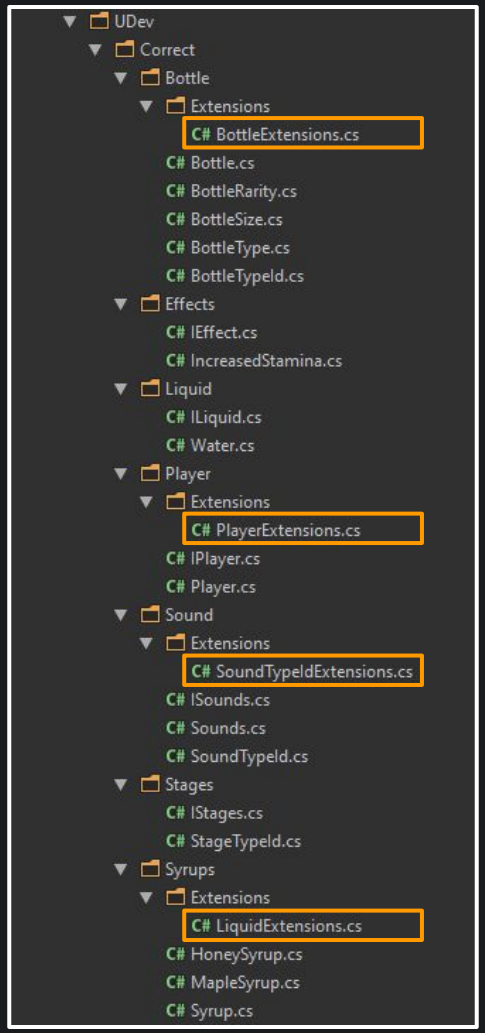

```
public class Bottle
{
  public BottleType Type{ ... }
  public Durability Durability{ ... }
  public BottleCap Cap{ ... }
}
```

Так как же все-таки открыть бутылку с водой?

Всё решает предметная область :-)







THANK YOU

TAKE THE WORLD