объектно-ориентированное программирование

основы

парадигмы

структурное программирование

```
0 VAR i
1 SET i 1
2 PRINT i
3 INC i
4 JIFLS i 10 2
```

```
for (var i = 1; i < 10; i++)
{
    Console.WriteLine(i);
}
```

парадигмы

процедурное программирование

```
var probability1 = Random.Shared.NextDouble();
var coefficient1 = probability1 > 0.5 ? 10d : 0;

var probability2 = Random.Shared.NextDouble();
var coefficient2 = probability2 > 0.5 ? 10d : 0;

var input = Console.ReadLine();
var value = input is null ? 0 : int.Parse(input);

var result = value * coefficient1 / coefficient2;

var message = $"result: {result}";
Console.WriteLine(message);
```

```
var coefficient1 = CalculateCoefficient();
var coefficient2 = CalculateCoefficient();

var value = ReadValue();

var result = value * coefficient1 / coefficient2;
OutputResult(result);

static double CalculateCoefficient()
{
    var probability = Random.Shared.NextDouble();
    return probability > 0.5 ? 10d : 0;
}

static int ReadValue()
{
    var input = Console.ReadLine();
    return input is null ? 0 : int.Parse(input);
}

static void OutputResult(double result)
{
    var message = $"result: {result}";
    Console.WriteLine(message);
}
```

```
static void MakePurchase(Account account, decimal cost)
{
    account.Balance -= cost;
}

static void ValidateBalance(Account account, decimal requiredBalance)
{
    if (account.Balance < requiredBalance)
        throw new InvalidOperationException($"Balance must be at least {requiredBalance}");
}

public class Account
{
    public decimal Balance;
}
```

```
static void MakePurchase(Account account, decimal cost)
{
    ValidateBalance(account, cost);
    account.Balance -= cost;
}

static void ValidateBalance(Account account, decimal requiredBalance)
{
    if (account.Balance < requiredBalance)
        throw new InvalidOperationException($"Balance must be at least {requiredBalance}");
}

public class Account
{
    public decimal Balance;
}
```

набор корректных состояний данных, определяемый набором бизнес-требований к этим данным

инвариант данных

концепции ООП

инкапсуляция

```
static void MoveBy(
    Point point,
    int deltaX,
    int deltaY)
{
    point.X += deltaX;
    point.Y += deltaY;
}

public class Point
{
    public int X;
    public int Y;
}
```

```
public class Point
{
    public int X;
    public int Y;

    public void MoveBy(int deltaX, int deltaY)
    {
        X += deltaX;
        Y += deltaY;
    }
}
```

концепции ООП

инкапсуляция

```
struct point
{
    int x;
    int y;
};

void move_by(point* p, int delta_x, int delta_y);
```

концепции ООП

сокрытие

```
public class Point
{
    public int X { get; private set; }
    public int Y { get; private set; }

    public void MoveBy(int deltaX, int deltaY)
    {
        ...
    }
}
```

концепции ООП

сокрытие

```
public class Point
{
    private int _x;
    private int _y;
}
```

```
public class Point
{
    ...

    public double DistanceFromStart => ...;

    ...
}
```

концепции ООП

композиция

агрегация

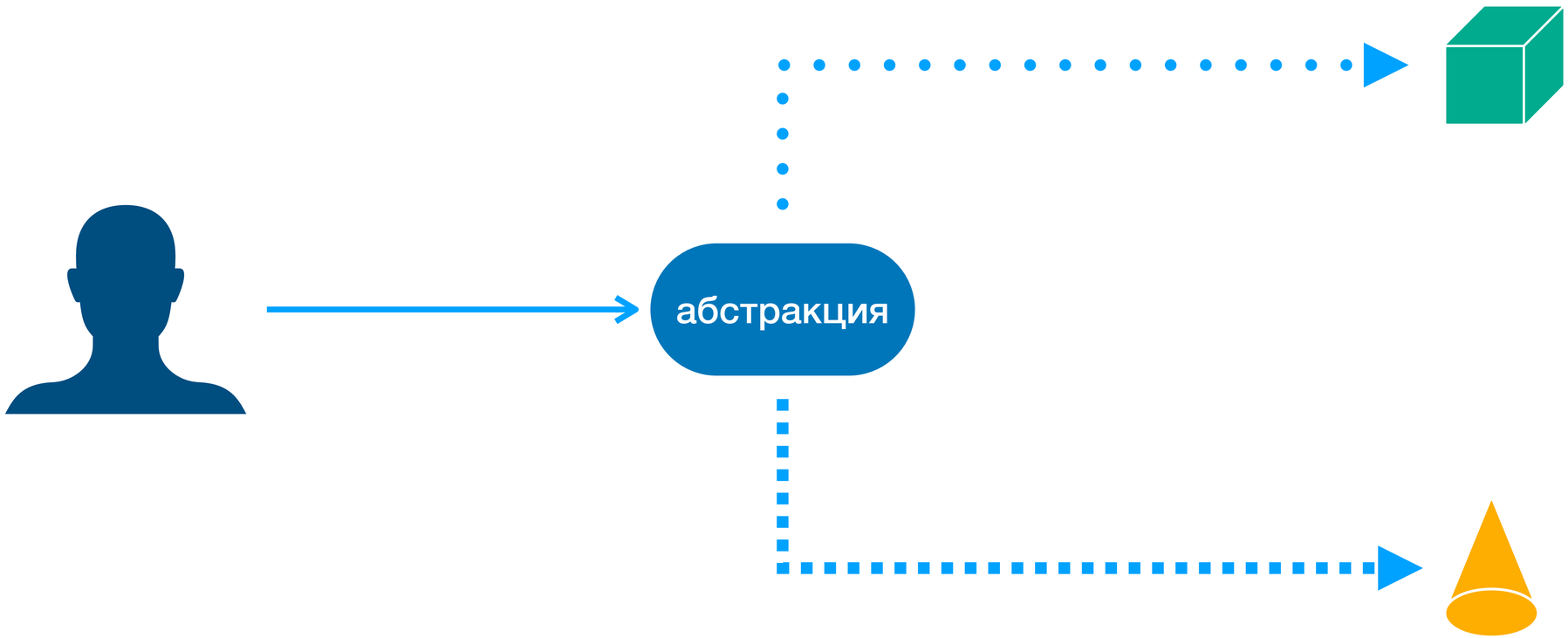
```
public class Point
{
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    ...
}


var point = new Point(420, 69);
```

ассоциация

```
public class RubicsCube
{
    private readonly Face[] _faces;

    public RubicsCube()
    {
        _faces = Enumerable
            .Range(0, 6)
            .Select(i ⇒ new Face(StickersFor(i)))
            .ToArray();
    }

    private static Sticker[] StickersFor(int faceIndex) { ... }
}
```

концепции ООП

полиморфизм

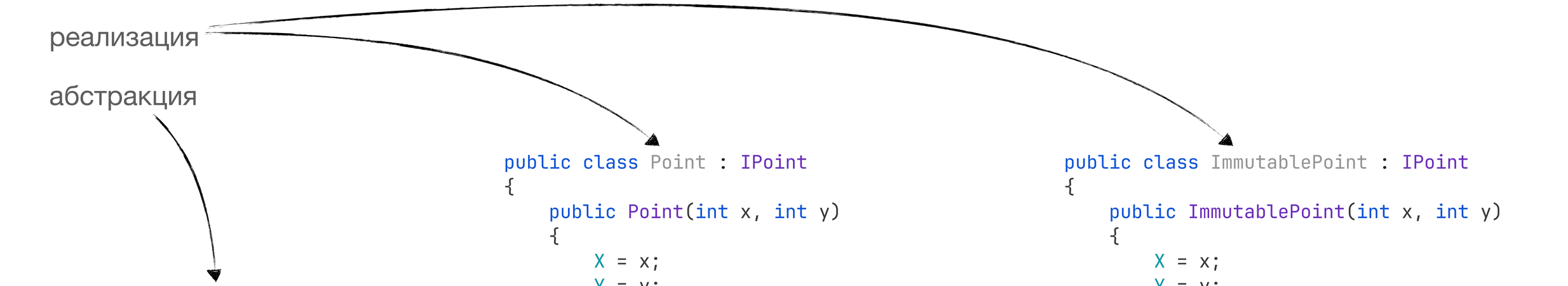


отделение абстракции от реализации, позволяющее пользователю прозрачно использовать различные реализации поведений

полиморфизм подтипов

концепции ООП

полиморфизм




```
public interface IPoint
{
    int X { get; }
    int Y { get; }

    double DistanceFromStart { get; }
}
```

```
public class Point : IPoint
{
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public int X { get; private set; }
    public int Y { get; private set; }

    public double DistanceFromStart => ...;

    public void MoveBy(int deltaX, int deltaY)
    {
        ...
    }
}
```

```
public class ImmutablePoint : IPoint
{
    public ImmutablePoint(int x, int y)
    {
        X = x;
        Y = y;

        DistanceFromStart = ...;
    }

    public int X { get; }
    public int Y { get; }

    public double DistanceFromStart { get; }
}
```

используются интерфейсы, в C# реализовывать интерфейсы могут как классы, так и структуры
говорят, что тип реализует интерфейс (класс `Point` реализует интерфейс `IPoint`)

реализация (наследование поведений)

концепции ООП

полиморфизм

наследование

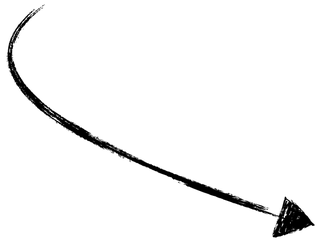
базовый класс

```
public class Animal
{
    public Animal(string name)
    {
        Name = name;
    }

    public string Name { get; }

    public virtual void Voice()
    {
        Console.WriteLine("I am an animal!");
    }
}
```

производный класс

```
public class Cat : Animal
{
    public Cat(string name) : base(name) { }

    public override void Voice()
    {
        Console.WriteLine("Meow!");
    }
}
```

используются классы, в C# одна структура не может быть унаследована от другой, либо от класса
говорят, что класс является наследником другого класса, либо же его подклассом (класс `Cat` является наследником класса `Animal`)



наследование (наследование реализаций)

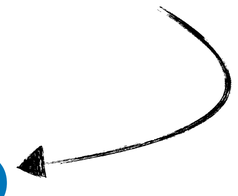
набор атрибутов и поведений, реализация и данные которого сокрыта от конечного пользователя объекта
абстракция, представляющая какой-то объект моделируемой предметной области

объект

наследование

best practices

применение наследования
для переиспользования
данных



ПЛОХО

```
public enum Suit
{
    Hearts,
    Diamonds,
    Clubs,
    Spades,
}

public enum CardValue
{
    Six,
    Seven,
    Eight,
    Nine,
    Ten,
    Jack,
    Queen,
    King,
    Ace,
}

public class Card
{
    public Card(Suit suit, CardValue value)
    {
        Suit = suit;
        Value = value;
    }

    public Suit Suit { get; }

    public CardValue Value { get; }
}
```

```
public class Card
{
    public Card(Suit suit, CardValue value) { ... }

    public Suit Suit { get; }

    public CardValue Value { get; }
}

public class Deck
{
    public Deck(IReadOnlyCollection<Card> cards) { ... }

    public IReadOnlyCollection<Card> Cards { get; }

    public void Shuffle()
    {
        ...
    }
}

public class Dealer : Deck
{
    public Dealer(IReadOnlyCollection<Card> cards) : base(cards) { }

    public void StartGame()
    {
        ...

        Shuffle();

        ...
    }
}
```

наследование

best practices

применение наследования
для достижения
полиморфизма

ХОРОШО

```
public record struct Coordinate(double X, double Y);

public abstract class Car
{
    public abstract void MoveTo(Coordinate coordinate);
}

public class Truck : Car
{
    public override void MoveTo(Coordinate coordinate)
    {
        Console.WriteLine($"Moving slowly to {coordinate}");
    }
}

public class SportCar : Car
{
    public override void MoveTo(Coordinate coordinate)
    {
        Console.WriteLine($"Rapidly moving to {coordinate}");
    }
}
```

ВЫВОДЫ

- парадигма ООП представляет собой концепцию объединения данных и логики, их обрабатывающей
- сокрытие принуждает пользователей использовать поведения соответствующие бизнес правилам
- локализация изменений данных позволяет упростить поддержание их инварианта