Шаблонни класове и методи — 12б, ООП

1. Увод

Шаблоните (Generics) са мощна функционалност в C#, която ни позволява да създаваме класове, методи и интерфейси, които работят с различни типове данни, като запазват типовата безопасност.

2. Какво са шаблоните?

Шаблоните позволяват:

- **Типова безопасност** - компилаторът проверява типовете
- **Преизползване на код** - един код за множество типове
- **Производителност** - без boxing/unboxing
- **Гъвкавост** - работа с различни типове данни
- **Читаемост** - по-ясен и понятен код

3. Шаблонни класове

Синтаксис на шаблонните класове:

- **<T>** - дефиниране на тип параметър
- **Множество параметри** - <T, U, V>

- **Ограничения** - where T : constraint
- **Използване** - ClassName<int> obj = new ClassName<int>();

Основен пример с шаблонен клас:

```
// Шаблонен клас за списък
class GenericList {
    private T[] items;
    private int count;
    private int capacity;

    public GenericList(int initialCapacity = 4) {
        capacity = initialCapacity;
        items = new T[capacity];
        count = 0;
    }

    public void Add(T item) {
        if (count >= capacity) {
            Resize();
        }
        items[count] = item;
        count++;
    }

    public T Get(int index) {
        if (index < 0 || index >= count) {
            throw new IndexOutOfRangeException("Индексът е извън
границите");
        }
        return items[index];
    }

    public void Set(int index, T item) {
        if (index < 0 || index >= count) {
            throw new IndexOutOfRangeException("Индексът е извън
границите");
        }
        items[index] = item;
    }

    public int Count {
```

```
        get { return count; }
    }

    public bool IsEmpty {
        get { return count == 0; }
    }

    private void Resize() {
        capacity *= 2;
        T[] newItems = new T[capacity];
        Array.Copy(items, newItems, count);
        items = newItems;
    }

    public void Display() {
        Console.WriteLine($"Списък с {count} елемента:");
        for (int i = 0; i < count; i++) {
            Console.WriteLine($"[{i}] = {items[i]}");
        }
    }
}

// Използване с различни типове
GenericList intList = new GenericList();
intList.Add(10);
intList.Add(20);
intList.Add(30);
intList.Display();

GenericList stringList = new GenericList();
stringList.Add("Здравей");
stringList.Add("Свят");
stringList.Display();

GenericList doubleList = new GenericList();
doubleList.Add(3.14);
doubleList.Add(2.71);
doubleList.Display();
```

4. Шаблонни методи

Характеристики на шаблонните методи:

- **Тип параметри** - <T> в декларацията
- **Автоматично определяне** - компилаторът може да определи типа
- **Ограничения** - where T : constraint
- **Преизползване** - един метод за множество типове

Пример с шаблонни методи:

```
class ArrayHelper {
    // Шаблонен метод за размяна на елементи
    public static void Swap(ref T a, ref T b) {
        T temp = a;
        a = b;
        b = temp;
    }

    // Шаблонен метод за намиране на максимален елемент
    public static T FindMax(T[] array) where T : IComparable {
        if (array == null || array.Length == 0) {
            throw new ArgumentException("Масивът е празен или null");
        }

        T max = array[0];
        for (int i = 1; i < array.Length; i++) {
            if (array[i].CompareTo(max) > 0) {
                max = array[i];
            }
        }
        return max;
    }

    // Шаблонен метод за търсене на елемент
    public static int FindIndex(T[] array, T item) where T : IEquatable {
        if (array == null) return -1;

        for (int i = 0; i < array.Length; i++) {
            if (array[i].Equals(item)) {
                return i;
            }
        }
        return -1;
    }
```

```
    // Шаблонен метод за обръщане на масив
    public static void Reverse(T[] array) {
        if (array == null) return;

        int left = 0;
        int right = array.Length - 1;

        while (left < right) {
            Swap(ref array[left], ref array[right]);
            left++;
            right--;
        }
    }

    // Шаблонен метод за копиране на масив
    public static T[] Copy(T[] source) {
        if (source == null) return null;

        T[] copy = new T[source.Length];
        Array.Copy(source, copy, source.Length);
        return copy;
    }

    // Шаблонен метод за показване на масив
    public static void Display(T[] array) {
        if (array == null) {
            Console.WriteLine("Масивът е null");
            return;
        }

        Console.WriteLine($"Масив с {array.Length} елемента:");
        for (int i = 0; i < array.Length; i++) {
            Console.WriteLine($"[{i}] = {array[i]}");
        }
    }
}

// Използване
int[] numbers = { 5, 2, 8, 1, 9 };
string[] words = { "ябълка", "банан", "портокал" };
double[] decimals = { 3.14, 2.71, 1.41 };

// Размяна на елементи
int a = 10, b = 20;
Console.WriteLine($"Преди: a = {a}, b = {b}");
ArrayHelper.Swap(ref a, ref b);
```

```
Console.WriteLine($"След: a = {a}, b = {b}");

// Намиране на максимален елемент
Console.WriteLine($"Максимално число: {ArrayHelper.FindMax(numbers)}");
Console.WriteLine($"Максимална дума: {ArrayHelper.FindMax(words)}");
Console.WriteLine($"Максимално десетично:
{ArrayHelper.FindMax(decimals)}");

// Търсене на елемент
int index = ArrayHelper.FindIndex(words, "банан");
Console.WriteLine($"Индекс на 'банан': {index}");

// Обръщане на масив
ArrayHelper.Display(numbers);
ArrayHelper.Reverse(numbers);
ArrayHelper.Display(numbers);
```

5. Ограничения на типовете (Constraints)

Видове ограничения:

- **where T : class** - T трябва да е референтен тип
- **where T : struct** - T трябва да е стойностен тип
- **where T : new()** - T трябва да има конструктор без параметри
- **where T : BaseClass** - T трябва да наследява BaseClass
- **where T : IInterface** - T трябва да имплементира IInterface

Пример с ограничения:

```
// Интерфейс за сравняване
interface IComparable {
    int CompareTo(T other);
}

// Базов клас за животни
```

```
class Animal {
    public string Name { get; set; }
    public int Age { get; set; }

    public Animal(string name, int age) {
        Name = name;
        Age = age;
    }
}

// Клас за кучета
class Dog : Animal, IComparable {
    public string Breed { get; set; }

    public Dog(string name, int age, string breed) : base(name, age) {
        Breed = breed;
    }

    public int CompareTo(Dog other) {
        return Age.CompareTo(other.Age);
    }

    public override string ToString() {
        return $"{Name} ({Breed}, {Age} години)";
    }
}

// Шаблонен клас с ограничения
class GenericContainer where T : class, new() {
    private T item;

    public GenericContainer() {
        item = new T(); // Възможно заради new() ограничението
    }

    public T Item {
        get { return item; }
        set { item = value; }
    }

    public void Reset() {
        item = new T();
    }
}

// Шаблонен клас за сравняване
class SortedList where T : IComparable {
```

```
    private List items;

    public SortedList() {
        items = new List();
    }

    public void Add(T item) {
        items.Add(item);
        items.Sort(); // Възможно заради IComparable ограничението
    }

    public T GetMin() {
        if (items.Count == 0) {
            throw new InvalidOperationException("Списъкът е празен");
        }
        return items[0];
    }

    public T GetMax() {
        if (items.Count == 0) {
            throw new InvalidOperationException("Списъкът е празен");
        }
        return items[items.Count - 1];
    }

    public void Display() {
        Console.WriteLine("Сортиран списък:");
        foreach (var item in items) {
            Console.WriteLine($"- {item}");
        }
    }
}

// Шаблонен метод с множество ограничения
class GenericHelper {
    public static T CreateAndInitialize() where T : class, new() {
        return new T();
    }

    public static void SortArray(T[] array) where T : IComparable {
        Array.Sort(array);
    }

    public static T FindItem(T[] array, T item) where T : IEquatable {
        foreach (var element in array) {
            if (element.Equals(item)) {
                return element;
```

```
            }
        }
        return default(T);
    }
}

// Използване
var container = new GenericContainer();
container.Item = new Dog("Рекс", 3, "Лабрадор");
Console.WriteLine($"Съдържание: {container.Item}");

var sortedDogs = new SortedList();
sortedDogs.Add(new Dog("Рекс", 3, "Лабрадор"));
sortedDogs.Add(new Dog("Бади", 1, "Джак Ръсел"));
sortedDogs.Add(new Dog("Ласи", 5, "Коли"));
sortedDogs.Display();

int[] numbers = { 5, 2, 8, 1, 9 };
GenericHelper.SortArray(numbers);
ArrayHelper.Display(numbers);
```

6. Шаблонни интерфейси

Характеристики на шаблонните интерфейси:

- **Тип параметри** - <T> в декларацията
- **Ограничения** - where T : constraint
- **Ковариантност** - out T за връщане на стойности
- **Контравариантност** - in T за приемане на параметри

Пример с шаблонни интерфейси:

```
// Шаблонен интерфейс за репозиторий
interface IRepository {
    void Add(T item);
    void Remove(T item);
```

```
    T GetById(int id);
    List GetAll();
    void Update(T item);
}

// Шаблонен интерфейс за сравняване
interface IComparer {
    int Compare(T x, T y);
}

// Шаблонен интерфейс за филтриране
interface IFilter {
    bool IsMatch(T item);
}

// Шаблонен интерфейс за сериализация
interface ISerializer {
    string Serialize(T obj);
    T Deserialize(string data);
}

// Конкретна имплементация за потребители
class User {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }

    public User(int id, string name, string email) {
        Id = id;
        Name = name;
        Email = email;
    }

    public override string ToString() {
        return $"User: {Name} ({Email})";
    }
}

// Имплементация на репозиторий за потребители
class UserRepository : IRepository {
    private List users;
    private int nextId;

    public UserRepository() {
        users = new List();
        nextId = 1;
    }
```

```
    public void Add(User user) {
        user.Id = nextId++;
        users.Add(user);
        Console.WriteLine($"Добавен потребител: {user}");
    }

    public void Remove(User user) {
        users.Remove(user);
        Console.WriteLine($"Премахнат потребител: {user}");
    }

    public User GetById(int id) {
        return users.FirstOrDefault(u => u.Id == id);
    }

    public List GetAll() {
        return new List(users);
    }

    public void Update(User user) {
        var existingUser = GetById(user.Id);
        if (existingUser != null) {
            existingUser.Name = user.Name;
            existingUser.Email = user.Email;
            Console.WriteLine($"Обновен потребител: {user}");
        }
    }
}

// Имплементация на сравняване за потребители
class UserNameComparer : IComparer {
    public int Compare(User x, User y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return string.Compare(x.Name, y.Name,
StringComparison.OrdinalIgnoreCase);
    }
}

// Имплементация на филтър за потребители
class UserEmailFilter : IFilter {
    private string domain;

    public UserEmailFilter(string domain) {
```

```
        this.domain = domain;
    }

    public bool IsMatch(User user) {
        return user.Email.EndsWith(domain);
    }
}

// Имплементация на сериализатор за потребители
class UserSerializer : ISerializer {
    public string Serialize(User user) {
        return $"{user.Id}|{user.Name}|{user.Email}";
    }

    public User Deserialize(string data) {
        var parts = data.Split('|');
        if (parts.Length != 3) {
            throw new ArgumentException("Невалиден формат на данните");
        }

        return new User(int.Parse(parts[0]), parts[1], parts[2]);
    }
}

// Използване
var userRepo = new UserRepository();
var user1 = new User(0, "Иван Петров", "ivan@example.com");
var user2 = new User(0, "Мария Георгиева", "maria@company.com");
var user3 = new User(0, "Петър Стоянов", "petar@example.com");

userRepo.Add(user1);
userRepo.Add(user2);
userRepo.Add(user3);

Console.WriteLine("\nВсички потребители:");
foreach (var user in userRepo.GetAll()) {
    Console.WriteLine(user);
}

// Филтриране по домейн
var emailFilter = new UserEmailFilter("@example.com");
var filteredUsers = userRepo.GetAll().Where(u => emailFilter.IsMatch(u));
Console.WriteLine("\nПотребители с @example.com:");
foreach (var user in filteredUsers) {
    Console.WriteLine(user);
}
```

```
// Сортиране по име
var nameComparer = new UserNameComparer();
var sortedUsers = userRepo.GetAll().OrderBy(u => u, nameComparer);
Console.WriteLine("\nСортирани потребители:");
foreach (var user in sortedUsers) {
    Console.WriteLine(user);
}

// Сериализация
var serializer = new UserSerializer();
string serialized = serializer.Serialize(user1);
Console.WriteLine($"\nСериализиран потребител: {serialized}");

var deserializedUser = serializer.Deserialize(serialized);
Console.WriteLine($"Възстановен потребител: {deserializedUser}");
```

7. Пълен пример - Система за управление на склад

Реален пример с шаблони:

```
// Шаблонен интерфейс за продукти
interface IProduct where T : IComparable {
    T Id { get; }
    string Name { get; }
    decimal Price { get; }
    int Quantity { get; set; }
    DateTime CreatedDate { get; }
}

// Базов клас за продукти
abstract class ProductBase : IProduct where T : IComparable {
    public T Id { get; }
    public string Name { get; }
    public decimal Price { get; }
    public int Quantity { get; set; }
    public DateTime CreatedDate { get; }

    protected ProductBase(T id, string name, decimal price, int quantity) {
        Id = id;
```

```
        Name = name;
        Price = price;
        Quantity = quantity;
        CreatedDate = DateTime.Now;
    }

    public abstract string GetCategory();
    public abstract bool IsAvailable();

    public virtual decimal GetTotalValue() {
        return Price * Quantity;
    }

    public override string ToString() {
        return $"{GetCategory()}: {Name} (ID: {Id}, Цена: {Price:C},
Количество: {Quantity})";
    }
}

// Конкретна имплементация за електроника
class Electronics : ProductBase where T : IComparable {
    public string Brand { get; }
    public string Model { get; }
    public int WarrantyMonths { get; }

    public Electronics(T id, string name, decimal price, int quantity,
                       string brand, string model, int warrantyMonths)
                       : base(id, name, price, quantity) {
        Brand = brand;
        Model = model;
        WarrantyMonths = warrantyMonths;
    }

    public override string GetCategory() {
        return "Електроника";
    }

    public override bool IsAvailable() {
        return Quantity > 0;
    }

    public override string ToString() {
        return base.ToString() + $" [Марка: {Brand}, Модел: {Model},
Гаранция: {WarrantyMonths} месеца]";
    }
}
```

```
// Конкретна имплементация за книги
class Book : ProductBase where T : IComparable {
    public string Author { get; }
    public string ISBN { get; }
    public int Pages { get; }

    public Book(T id, string name, decimal price, int quantity,
                string author, string isbn, int pages)
                : base(id, name, price, quantity) {
        Author = author;
        ISBN = isbn;
        Pages = pages;
    }

    public override string GetCategory() {
        return "Книги";
    }

    public override bool IsAvailable() {
        return Quantity > 0;
    }

    public override string ToString() {
        return base.ToString() + $" [Автор: {Author}, ISBN: {ISBN},
Страници: {Pages}]";
    }
}

// Шаблонен клас за склад
class Warehouse where T : IProduct where U : IComparable {
    private Dictionary products;
    private List productIds;

    public Warehouse() {
        products = new Dictionary();
        productIds = new List();
    }

    public void AddProduct(T product) {
        if (products.ContainsKey(product.Id)) {
            throw new ArgumentException($"Продукт с ID {product.Id} вече
съществува");
        }

        products[product.Id] = product;
        productIds.Add(product.Id);
        Console.WriteLine($"Добавен продукт: {product}");
```

```
    }

    public void RemoveProduct(U id) {
        if (!products.ContainsKey(id)) {
            throw new ArgumentException($"Продукт с ID {id} не
съществува");
        }

        var product = products[id];
        products.Remove(id);
        productIds.Remove(id);
        Console.WriteLine($"Премахнат продукт: {product}");
    }

    public T GetProduct(U id) {
        if (!products.ContainsKey(id)) {
            throw new ArgumentException($"Продукт с ID {id} не
съществува");
        }
        return products[id];
    }

    public List GetAllProducts() {
        return productIds.Select(id => products[id]).ToList();
    }

    public List GetProductsByCategory(string category) {
        return products.Values.Where(p => p.GetCategory() ==
category).ToList();
    }

    public List GetAvailableProducts() {
        return products.Values.Where(p => p.IsAvailable()).ToList();
    }

    public decimal GetTotalValue() {
        return products.Values.Sum(p => p.GetTotalValue());
    }

    public void UpdateQuantity(U id, int newQuantity) {
        if (!products.ContainsKey(id)) {
            throw new ArgumentException($"Продукт с ID {id} не
съществува");
        }

        var product = products[id];
        product.Quantity = newQuantity;
```

```
        Console.WriteLine($"Обновено количество за {product.Name}: 
{newQuantity}");
    }

    public void DisplayInventory() {
        Console.WriteLine($"\n=== Складова наличност ===");
        Console.WriteLine($"Общо продукти: {products.Count}");
        Console.WriteLine($"Обща стойност: {GetTotalValue():C}");
        Console.WriteLine();

        foreach (var product in products.Values) {
            Console.WriteLine(product);
        }
    }
}

// Шаблонен клас за търсене
class ProductSearch where T : IProduct where U : IComparable {
    public List SearchByName(List products, string name) {
        return products.Where(p => 
p.Name.ToLower().Contains(name.ToLower())).ToList();
    }

    public List SearchByPriceRange(List products, decimal minPrice, decimal 
maxPrice) {
        return products.Where(p => p.Price >= minPrice && p.Price <= 
maxPrice).ToList();
    }

    public List SearchByCategory(List products, string category) {
        return products.Where(p => p.GetCategory() == category).ToList();
    }

    public List SearchAvailable(List products) {
        return products.Where(p => p.IsAvailable()).ToList();
    }
}

// Използване
var warehouse = new Warehouse, int>();
var bookWarehouse = new Warehouse, string>();

// Добавяне на електроника
var laptop = new Electronics(1, "Laptop Dell", 1500, 5, "Dell", "Inspiron 
15", 24);
var phone = new Electronics(2, "iPhone 15", 1200, 10, "Apple", "iPhone 15", 
12);
```

```
var tablet = new Electronics(3, "iPad Air", 800, 3, "Apple", "iPad Air",
12);

warehouse.AddProduct(laptop);
warehouse.AddProduct(phone);
warehouse.AddProduct(tablet);

// Добавяне на книги
var book1 = new Book("B001", "C# Programming", 50, 20, "Иван Петров", "978-
1234567890", 400);
var book2 = new Book("B002", "OOP Principles", 45, 15, "Мария Георгиева",
"978-0987654321", 350);

bookWarehouse.AddProduct(book1);
bookWarehouse.AddProduct(book2);

// Показване на складовете
warehouse.DisplayInventory();
bookWarehouse.DisplayInventory();

// Търсене
var search = new ProductSearch, int>();
var appleProducts = search.SearchByName(warehouse.GetAllProducts(),
"Apple");
Console.WriteLine("\nApple продукти:");
foreach (var product in appleProducts) {
    Console.WriteLine(product);
}

var expensiveProducts =
search.SearchByPriceRange(warehouse.GetAllProducts(), 1000, 2000);
Console.WriteLine("\nСкъпи продукти (1000-2000 лв):");
foreach (var product in expensiveProducts) {
    Console.WriteLine(product);
}
```

8. Предимства на шаблоните

Ключови предимства:

- **Типова безопасност** - компилаторът проверява типовете

- **Производителност** - без boxing/unboxing
- **Преизползване на код** - един код за множество типове
- **Читаемост** - по-ясен и понятен код
- **Гъвкавост** - работа с различни типове данни

9. Практически задачи

Задачи за упражнение:

- **Създай шаблонен клас Stack<T>** с методи Push, Pop, Peek
- **Имплементирай шаблонен клас Queue<T>** с Enqueue, Dequeue
- **Направи шаблонен клас Dictionary<K, V>** с Add, Remove, Get
- **Създай шаблонен клас Calculator<T>** с математически операции
- **Имплементирай шаблонен клас Cache<T>** с TTL функционалност

10. Заключение

Шаблоните са мощна функционалност, която ни позволява да създаваме гъвкав, типобезопасен и преизползваем код, който работи с различни типове данни.

Ключови принципи:

- **Типова безопасност** - компилаторът проверява типовете
- **Преизползване** - един код за множество типове
- **Производителност** - без допълнителни разходи
- **Гъвкавост** - работа с различни типове данни

- **Ограничения** - контрол върху използваните типове