A black background with a black square

Description automatically generated with medium confidence **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КОМПЮТЪРНО И СОФТУЕРНО ИНЖЕНЕРСТВО

**Програмни Среди:**

Курсова работа:

„Дигитализация на структура на въпрос“

**Реализация**

|  |  |  |
| --- | --- | --- |
| Съставил: | Факултетен номер: | Група: |
| Ангел Любомиров Стойнов | 121222150 | 40 |

1. Интерфейсни типове в програмирането

Интерфейсите в програмирането са абстрактни типове, които определят набор от методи, свойства, индексатори и събития, които клас или структура трябва да имплементира. Те са ключов елемент в концепцията за полиморфизъм и абстракция, което позволява на програмистите да пишат по-модулни и адаптивни програми. Интерфейсите не съдържат реализация на дефинираните членове – те само задават формата. Реализацията на тези методи и свойства се осъществява от конкретните типове, които имплементират интерфейса.

1. Интерфейс в .Net

2.1Използване на интерфейси със стойностни типове

В езици като C#, стойностните типове (пр. структури) се обработват малко по-различно от референтните типове (като класове). Структурите се предават по стойност, което означава, че всяко предаване или присвояване на структура води до създаването на ново копие от данните. Това може да намали натоварването върху управлението на паметта, тъй като обекти на стойностни типове не изискват събиране на боклука(garbage collection) и се изтриват, когато излязат извън обсега.

Използването на интерфейси със стойностни типове позволяват на програмистите да използват полиморфизъм със структура. Това означава, че могат да се създават функции или методи, които работят с интерфейси и следователно могат да приемат всякакви стойностни типове, които имплементират този интерфейс, увеличавайки повторната употреба на кода.

**Предизвикателства при използване на интерфейси със стойностни типове:**1. **Boxing and Unboxing:** Когато стойностни типове се използват като интерфейси, той трябва да бъде „опакован“ в референтния тип, което може да води до ненужни разходи на памет и време за изпълнение. Това е така, защото структурите, които обикновено се съхраняват на стека трябва да бъдат преместени в управляема купчина, за да бъдат обработени като обекти.  
2. **Производителност:** Преместването между стека и купчината и необходимостта от събиране на боклука (garbage collection) заопакованите обекти могат да доведат до снижаване на производителността особено в приложения, които изискват висока производителност и ефективност.

1. **Мутационни ефекти:** Когато стойностния тип е опакован, всяка модификация за него върху интерфейса не се отразява на оригиналния обект, защото операциите се извършват върху копие. Това може да доведе до бъгове и неконсистентност в данните, ако програмистите не са наясно с тези поведения.
2. Интерфейсни типове в програмирането

1.1 **Дефиниция и предназначение:** Интерфейсите в програмирането са абстрактни типове, които дефинират набор от методи, свойства, индексатори и събития, които трябва да бъдат имплементирани от класове или структури.

1.2 **Значение и ползи от интерфейсите:**

* Поддържат абстракция и полиморфизъм, позволявайки по-лесно поддържане и адаптиране на кода.
* Улесняват създаването на модулни и гъвкави приложения.
* Подобряват преизползваемостта и поддържат по-добра организация на кода.

1.3 **Характеристики:**

* Интерфейсите не съдържат реализация на дефинираните членове.
* Реализацията се осигурява от конкретните типове (класове или структури).
* Един тип може да имплементира множество интерфейси, което позволява голяма гъвкавост.

1. **Интерфейс в .NET**

2.1 **Общи характеристики в .NET:**

* Интерфейсите могат да бъдат имплементирани както от референтни (класове), така и от стойностни типове (структури).
* Дефинират начина, който трябва да се спази от имплементиращите го типове.

2.2 **Използване на интерфейси със стойностни типове (структури)**

2.2.1 **Предимства**

* Позволяват използване на полиморфизъм със структури.
* Увеличават възможността за преизползване на кода, тъй като методите могат да работят с различни структури, имплементиращи общ интерфейс.
* По-малко натоварване на паметта при стандартно използване, тъй като не изискват garbage collection и се освобождават веднага след излизане извън обсег.

**2.2.2 Предизвикателства и недостатъци**

**А. Boxing и Unboxing**

* При използване на структури като интерфейси, стойностните типове трябва да бъдат „опаковани“ (boxing) в референтни типове, което води до допълнителна консумация на памет и процесорно време.
* Това се случва, защото структурите, които обикновено са на стека, се преместват в heap, когато са използвани чрез интерфейс.

**Б. Производителност**

* Преместването между стека и heap води до допълнителни разходи.
* Необходимостта от garbage collection за опакованите обекти може да понижи производителността, особено в системи, изискващи висока ефективност.

**В. Мутационни ефекти**

* Модификации върху опакован стойностен тип чрез интерфейс не засягат оригиналния обект, а само неговото копие.
* Това може да доведе до бъгове и неконсистентност, ако не е правилно разбрано и предвидено от разработчиците.

2.3 Добавени от мен подточки за интерфейсни типове в .NET

* Интерфейсите поддържат множествено наследяване на типове (за разлика от класовете в .NET).
* От C# 8.0 интерфейсите могат да имат default методи с реализации, което подобрява възможностите за еволюция на интерфейсите без нарушаване на съществуващи имплементации.
* Интерфейсите са важен инструмент в Dependency Injection (DI), което подобрява тестируемостта и поддръжката на кода.

Какво е интерфейсен тип? - Интерфейсите са референтен тип данни, които не могат да бъдат инстанцирани директно и да съдържат имплементиращ код; Интерфейсите са средство за постигане на множествено наследяване.

· Създаване на интерфейс

- Интерфейс в С# се създава с помощта на ключовата дума interface.; **Счита се за добра практика името на интерфейса да започва с буквата "I"**; Членовете на интерфейса по подразбиране са абстрактни и публични; Интерфейсите могат да съдържат свойства и методи, но не и полета/променливи; Интерфейсът не може да съдържа конструктор и деструктор, тъй като не може да се използва за създаване на обекти; В интерфейсите не може да има константи.

Имплементация на интерфейс

За имплементация на интерфейс в С# се използва символа ":", последван от името на интерфейса.; Интерфейсите се реализират от класове или структури, които имплементират всички дефинирани в тях членове.; Един клас може да имплементира множество интерфейси.; Не е необходимо да използвате ключовата дума override при внедряване на интерфейс.;

• Предимства и недостатъци

Предимства

Множествено наследяване - С# не поддържа множествено

наследяване за класове, но един клас може да реализира множество интерфейси.

■ Разширяемост на кода Интерфейсите позволяват да се добавя нова функционалност към съществуващи класове, без да се променя изходния код. Може да се създават нови интерфейси и да се внедряват в съществуващите класове, като подобряват възможностите на тези класове.

Недостатъци

Без имплементация - Имплементацията на интерфейс изисква предоставяне на конкретни реализации за всички членове, което може да доведе до излишен код в класове, които прилагат множество интерфейси.

Interface Pollution - Когато един клас имплементира множество интерфейси, може да се наложи да имплементира много членове от различни интерфейси. Това може да доведе до ситуация, известна като "Interface Pollution“, при която интерфейсът на класа става претрупан с несвързани членове.

Основни интерфейсни типове:

IComparable

Често пъти освен за равенство е необходимо обектите да се сравняват спрямо някаква подредба (например лексикографска за низове или по големина за числови типове). В .NET Framework типовете, които могат да бъдат сравнявани един с друг, трябва да имплементират интерфейса System. IComparable. Интерфейсът дефинира един-единствен метод - Compare To(object). IComparable се използва от .NET Framework при сортиране на масиви и колекции и при някои други операции, изискващи сравнение по големина.

IComparable е имплементиран от много системни .NET типове, като например от примитивните стойностни типове System.Char, System.Int32, System.Single, System.Double, от символните низове (System.String) и от изброените типове (System. Enum).

о IEnumerable

Интерфейсът System.IEnumerable се имплементира от колекции и други типове, които поддържат операцията "обхождане на елементите им в някакъв ред". Този интерфейс дефинира само един метод - методът GetEnumerator(). Той връща итератор (инстанция на IEnumerator) за обхождане на елементите на дадения обект. Обектите, поддържащи IEnumerable интерфейса, могат да се използват от конструкцията foreach в С# за обхождане на всичките им елементи. Интерфейсът IEnumerable e реализиран от много системни .NET типове, като System.Array, System. String, ArrayList, Hashtable, Stack, Queue, SortedList и др. с цел да се улесни работата с тях.

IEnumerator

Интерфейсът System.IEnumerator имплементира обхождане на всички елементи на колекции и други типове. Той реализира прост итератор чрез следните методи и свойства:

- Свойство Current - връща текущия елемент.

Метод bool MoveNext() - преминава към следващия елемент и връща true, ако той е валиден.

- Метод Reset() - премества итератора непосредствено преди първия елемент (установява го в начално състояние).

1. Интерфейсни типове в програмирането

1.1 Какво е интерфейсен тип?

* Интерфейсите са референтни типове данни, които не могат да бъдат инстанцирани директно.
* Те не съдържат реализация на методи, свойства и събития.
* Интерфейсите предоставят средство за постигане на множествено наследяване.

1.2 Създаване на интерфейс в C#

* Интерфейс се създава чрез ключовата дума interface.
* Добра практика е името на интерфейса да започва с "I".
* Членовете на интерфейса са по подразбиране абстрактни и публични.
* Интерфейсите могат да съдържат методи и свойства, но не могат да съдържат полета (променливи), константи, конструктори или деструктори.

1.3 Имплементация на интерфейс

* В C# интерфейс се имплементира с помощта на символа :, последван от името на интерфейса.
* Интерфейсите могат да бъдат имплементирани както от класове, така и от структури.
* Един клас може да имплементира множество интерфейси.
* Не е необходима ключовата дума override при внедряване на интерфейс.

1.4 Предимства и недостатъци на интерфейсите

Предимства:

* Позволяват множествено наследяване, което не е възможно при класовете.
* Осигуряват разширяемост на кода, като позволяват добавяне на нова функционалност към съществуващи класове без промяна на оригиналния код.

Недостатъци:

* Изискват имплементация на всички членове, което може да доведе до излишен код при класове, имплементиращи множество интерфейси.
* Риск от „Interface Pollution“ – претоварване на интерфейса на класа с несвързани членове от множество интерфейси.

1. Основни интерфейсни типове в .NET

2.1 IComparable

* Интерфейс за сравнение на обекти, дефиниращ метода CompareTo(object).
* Използва се при сортиране на масиви, колекции и операции, изискващи сравнение.
* Имплементиран от типове като System.Char, System.Int32, System.Double, System.String, и System.Enum.

2.2 IEnumerable

* Интерфейс, който дефинира метод GetEnumerator(), връщащ итератор за обхождане.
* Позволява използването на конструкцията foreach.
* Имплементиран от типове като System.Array, System.String, ArrayList, Hashtable, Stack, Queue, и SortedList.

2.3 IEnumerator

* Интерфейс, който дефинира итератор за обхождане на колекции с:
  + Свойство Current – връща текущия елемент.
  + Метод MoveNext() – преминава към следващия елемент.
  + Метод Reset() – връща итератора в начално състояние.

1. Използване на интерфейси със стойностни типове (структури)

3.1 Предимства

* Позволяват полиморфизъм и гъвкавост при структури.
* Намалено натоварване на паметта, защото не изискват garbage collection.

3.2 Предизвикателства и недостатъци

* **Boxing и Unboxing:** Преобразуването на стойностен тип към интерфейс (референтен тип) изисква "опаковане", водещо до повишени разходи за памет и производителност.
* **Производителност:** Допълнителни разходи при преместване между стека и купчината.
* **Мутационни ефекти:** Промени върху опаковани стойностни типове чрез интерфейс не влияят върху оригиналните обекти, което може да предизвика неочаквано поведение и бъгове.