Изпитна тема № 3: **Представяне на обекти от реалния свят с програмен код**

*Понятия: Клас, конструктор, поле и свойство. Създаване на обекти от клас. Функции/методи в класовете, тип и стойност на връщане, параметри и видове параметри на функция/метод – употреба. Необходимост от употреба на ключовата дума this. Енкапсулация на данни в класовете, методи за достъп и промяна на полета (getters/setters), модификатори за достъп. Статични полета и методи в класовете. Създаване и/или поправка/допълване на вече съществуващи компютърни програми, описващи обекти от реалния свят със средствата на програмен език*

**Обекти** са понятия в някоя предметна област, които се налага да моделираме и използваме в компютърна програма. Примери за абстрактни обекти са структурите от данни стек, опашка, списък и дърво.

В обектите от реалния свят (също и в абстрактните обекти) могат да се отделят следните две групи техни характеристики:

-     Състояния (states) – това са характеристики на обекта, които по някакъв начин го определят и описват по принцип или в конкретен момент.

-     Поведения (behaviors) – това са специфични характерни действия, които обектът може да извършва.

Обектите в ООП обединяват данни и средствата за тяхната обработка в едно цяло. Те съответстват на обектите от реалния свят и съдържат в себе си данни и действия:

-     Член-данни (data members) – представляват про­менливи, вградени в обектите, които описват състоянията им.

-     Методи (methods) – вече сме ги разглеждали в детайли. Те са инструментът за изграждане на поведението на обектите.

**Класът** дефинира характеристиките на даден обект (които ще наричаме атрибути) и неговото поведение (действията, които обектът може да извършва). Атрибутите на класа се дефинират като собствени променливи в тялото му (наречени член-променливи). Поведението на обектите се моделира чрез дефиниция на методи в класовете.

Класът в C# се дефинира чрез ключовата дума class, последвана от идентификатор (име) на класа и съвкупност от член-данни и методи, обособени в собствен блок код.

Класовете в C# могат да съдържат следните елементи:

- Полета (fields) – член-променливи от определен тип;

- Свойства (properties) – това са специален вид елементи, които разширяват функционалността на полетата като дават възможност за допълнителна обработка на данните при извличането и записва­нето им в полетата от класа.

- Методи – реализират манипулацията на данните.

Декларирането на клас следва строго определени правила (синтаксис):

[<access\_modifier>] class <class\_name>

Когато декларираме клас, задължително трябва да използваме ключовата дума class. След нея трябва да стои името на класа <class\_name>.

Освен ключовата дума class и името на класа, в декларацията на класа могат да бъдат използвани някои модификатори

По подобие на методите, след декларацията на класа следва неговото тяло, т.е. частта от класа, в която се съдържа програмния код:

[<access\_modifier>] class <class\_name>

{

// ... Class body – the code of the class goes here ...

}

Тялото на класа започва с отваряща фигурна скоба "{" и завършва със затваряща – "}". Класът винаги трябва да има тяло.

Деклариране на полета в даден клас

До момента сме се сблъсквали само с два типа променливи, в зависимост от това къде са декларирани:

1. Локални променливи – това са променливите, които са дефини­ра­ни в тялото на някой метод (или блок).

2. Параметри – това са променливите в списъка с параметри, които един метод може да има.

В C# съществува и трети вид променливи, наречени полета (fields) или член-променливи на класа (instance variables).

Полетата се декларират в тялото на класа, но извън тялото на метод, конструктор или блок.

Когато декларираме едно поле е възможно едновременно с неговата декларация да му дадем първоначална стойност. Начинът, по който става това, е същият както при инициализацията (даването на стойност) на обикновена локална променлива:

[<modifiers>] <field\_type> <field\_name> = <initial\_value>;

При създаване на обект всички негови полета се инициа­лизират с подразбиращите се стойности за типа им, освен ако изрично не бъдат инициализирани.

За разлика от полетата, локалните променливи, не биват инициализирани с подразбираща се стойност при тяхното деклариране.

Създаването на обекти от предварително дефинирани класове по време на изпълнението на програмата става чрез оператора new. Новосъздаденият обект обикновено се присвоява на променлива от тип, съвпадащ с класа на обекта. При това присвояване същинският обект не се копира, а в променливата се записва само референция към новосъздадения обект (неговият адрес в паметта). При създаването на обект с оператора new се случват две неща: заделя се памет за този обект и се извършва начална инициализация на член-данните му. Инициализацията се осъществява от специален метод на класа, наречен конструктор. Важна особеност на работата с обекти в C# e, че обикновено няма нужда от ръчното им разрушаване и освобождаване на паметта, заета от тях. Това е възможно поради вградената в .NET CLR система за почистване на паметта (garbage collector), която се грижи за освобождаването на неизползвани обекти вместо нас. Обектите, към които в даден момент вече няма референция в програмата, автоматично се унищожават и паметта, която заемат се освобождава. По този начин се предотвратяват много потенциални бъгове и проблеми.

Достъпът до полетата и свойствата (properties) на даден обект става чрез оператора . (точка), поставен между името на обекта и името на полето (или свойството). Операторът . не е необходим в случай, че достъпваме поле или свойство на даден клас в тялото на метод от същия клас. Можем да достъпваме полетата и свойствата или с цел да извлечем данните от тях, или с цел да запишем нови данни. В случай на свойство, достъпът се реализира по абсолютно същия начин както и при поле – C# ни предоставя тази възможност. Това се постига чрез двете специални ключови думи get и set в дефиницията на свойството, които извършват съответно извличането на стойността на свойството и присвояването на нова стойност.

Извикването на методите на даден обект става отново чрез оператора . (точка). Операторът точка не е нужен единствено, в случай че съответ­ният метод се извиква в тялото на друг метод от същия клас.

Сега е моментът да споменем факта, че методите на класовете имат модификатори за достъп public, private или protected, чрез които възможността за извикването им може да се ограничава.

**Статични полета и методи**

В ООП има специална категория полета и методи, които се асоциират с тип данни (клас), а не с конкретна негова инстанция (обект). Наричаме ги статични членове (static members), защото са независими от конкретните обекти. Нещо повече, те се използват без да има създадена инстанция на класа, в който са дефинирани. Те могат да бъдат полета, методи и конструктори.

Статично поле или метод в даден клас се дефинира чрез ключовата дума static, поставена преди типа на полето или типа на връщаната стойност на метода. При дефинирането на статичен конструктор думата static се поставя преди името на конструктора.

**Модификатори и нива на достъп (видимост)-** модификатор наричаме ключова дума с помощта, на която даваме допълнителна информация на компилатора за кода, за който се отнася модификаторът.

В C# има четири модификатора за достъп. Те са public, private, protected и internal. Модификатори за достъп могат да се използват само пред следните елементи на класа: декларация, полета, свойства и методи на класа.

Използвайки модификатора public, ние указваме на компилатора, че елементът, пред който е поставен, може да бъде достъпен от всеки друг клас, независимо дали е от текущия проект, от текущото пространство от имена или извън тях. Нивото на достъп public определя липса на ограничения върху видимостта, най-малко рестриктивното от всички нива на достъп в C#.

Нивото на достъп private, което налага най-голяма рестрикция на види­мостта на класа и елементите му. Модификаторът private служи за индикация, че елементът, за който се отнася, не може да бъде достъпван от никой друг клас (освен от класа, в който е дефиниран), дори този клас да се намира в същото пространство от имена. Това ниво на достъп се използва по подразбиране, т.е. се прилага, когато липсва модификатор за достъп пред съответния елемент на класа.

Модификаторът internal се използва, за да се ограничи достъпът до елемента само от файлове от същото асембли, т.е. същия проект във Visual Studio. Когато във Visual Studio направим няколко проекта, класовете от тях ще се компилират в различни асемблита.

Асембли (assembly) е колекция от типове и ресурси, която формира логическа единица функционалност. Всички типове в C# и изобщо в .NET Framework могат да съществуват само в асемблита. При всяка компилация на .NET приложение се създава асембли. То се съхранява като файл с разширение .exe или .dll.

**Ключовата дума this** в C# дава достъп до референцията към текущия обект, когато се използва от метод в даден клас. Това е обектът, чийто метод или конструктор бива извикван. Можем да я разглеждаме като указател (референция), дадена ни априори от създателите на езика, с която да достъпваме елементите (полета, методи, конструктори) на собствения ни клас:

* this е препратка към текущия обект
* this може да сочи към променлива, която е инстанция (представител) на текущия клас
* this може да се предава като аргумент в метод или като извикване на конструктор
* this може да се върне като стойност на метод
* this може да извика метод на текущия клас
* this може да извиква конструктор на текущия клас

Деклариране на методи в даден клас

Декларирането на методи, както знаем, става по следния начин:

|  |
| --- |
| // Method definition  **[<modifiers>] [<return\_type>] <method\_name>([<parameters\_list>])**  **{**        // ... Method’s body ...  **[<return\_statement>];**  **}** |

Задължителните елементи при декларирането на метода са типът на връ­ща­ната стойност <return\_type>, името на метода <method\_name> и отваря­щата и затварящата кръгли скоби – "(" и ")".

Списъкът от параметри <params\_list> не е задължителен. Използваме го да подаваме някакви данни на метода, който декларираме, ако той има нужда.

Знаем, че ако типът на връщаната стойност <return\_type> е void, тогава <return\_statement> може да участва само с оператора return без аргумент, с цел пре­кра­тя­ване действието на метода. Ако <return\_type> е различен от void, методът задължително трябва да връща резултат чрез ключовата ду­ма return с аргумент, който е от тип <return\_type> или съвместим с не­го.

Работата, която методът трябва да свърши, се намира в тялото му, заградена от фигурни скоби – "{" и "}".

**Конструкторъ**т е специален метод на класа, който се извиква автоматично при създаването на обект от този клас и извършва инициализация на данните му (това е неговото основно предназначение). Конструкторът няма тип на връщана стойност и неговото име не е произволно, а задължително съвпада с името на класа. Конструкторът може да бъде със или без параметри.

Конструктор без параметри наричаме още конструктор по подразбиране (default constructor). Езикът C# допуска да няма изрично дефиниран конструктор в класа. В този случай, компилаторът създава автоматично празен конструктор по подразбиране, който занулява всички полета на класа.

Конструкторът може да приема параметри, както всеки друг метод. Всеки клас може да има произволен брой конструктори с единственото ограни­чение, че броят и типът на параметрите им трябва да бъдат различни. При създаването на обект от този клас се извиква точно един от дефинираните конструктори.

Извикване на конструктор

Единственият начин да извикаме един конструктор в C# е чрез ключовата дума new. Тя заделя памет за новия обект (в стека или в хийпа според това дали обектът е стойностен или референтен тип), занулява полетата му, извиква конструктора му (или веригата конструктори, образувана при наследяване) и накрая връща референция към новозаделения обект.

Деклариране на конструктор

Формално, декларацията на конструктора изглежда по следния начин:

|  |
| --- |
| **[<modifiers>] <class\_name>([<parameters\_list>])** |

Както вече знаем, конструПкторите приличат на методи, но нямат тип на връщана стойност (затова ги нарекохме псевдометоди).

Инициализация на полета в конструктора

При създаването на нов обект и извикването на конструктор, се заделя памет за нестатичните полетата на обекта от дадения клас и те се инициализират със стойностите по подразбиране за техния тип

Освен това, чрез конструкторите най-често инициализираме полетата на класа, със стойности зададени от нас, а не с подразбиращите се за типа.

Свойства (Properties)

В света на обектно-ориентираното програмиране съществува елемент на класовете, наречен свойство (property), който е нещо средно между поле и метод и служи за по-добра защита на състоянието в класа.

Свойства – капсулация на достъпа до полетата

Основната цел на свойствата е да осигуряват капсулация на състоянието на класа, в който са декларирани, т.е. да го защитят от попадане в невалидни състояния.

Капсулация (encapsulation) наричаме скриването на физическото представяне на данните в един клас, така че, ако в последствие променим това представяне, това да не рефлектира върху останалите класове, които използват този клас.

Чрез синтаксиса на C#, това се реализира като декларираме полета (физи­чес­кото представяне на данните) с възможно най-ограничено ниво на видимост (най-често с модификатор private) и декларираме достъпът до тези полета (четене и модифициране) да може да се осъществява единствено чрез специални методи за достъп (accessor methods).

**Деклариране на свойства в C#**

За да декларираме едно свойство в C#, трябва да декларираме методи за достъп (за четене и промяна) на съответното свойство и да решим по какъв начин ще съхраняваме ин­фор­ма­цията за това свойство в класа.

Преди да декларираме методите обаче, трябва да декларираме самото свойството в класа. Формално декларацията на свойствата изглежда по следния начин:

|  |
| --- |
| [<modifiers>] <property\_type> <property\_name> |

С <modifiers> сме означили, както модифика­то­ри­те за достъп, така и други модификатори (например static, който ще разгледаме в [следва­щата секция на главата](https://introprogramming.info/intro-csharp-book/read-online/glava14-definirane-na-klasove/#_%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B8_%D0%BA%D0%BB%D0%B0%D1%81%D0%BE%D0%B2%D0%B5_(static)). Те не са задължи­тел­­на част от декларацията на едно поле.

Типа на свойството <property\_type> задава типа на стойностите на свойството. Може да бъде както примитивен тип (например int), така и референтен (например масив).

Съответно, <property\_name> е името на свойството. То трябва да започва с главна буква и да удовлетворява правилото PascalCase, т.е. всяка нова дума, която се долепя в задната част на името на свойството, започва с главна буква.

**Тяло на свойство**

Подобно на класа и методите, свойствата в С# имат тяло, където се декларират методите за достъп до свойството (accessors).

|  |
| --- |
| [<modifiers>] <property\_type> <property\_name>  {        // ... Property's accessors methods go here  } |

Тялото на свойството започва с отваряща фигурна скоба "{" и завършва със затваряща – "}". Свойствата винаги трябва да имат тяло.

**Метод за четене на стойността на свойство (getter)**

Както обяснихме, декларацията на метод за четене на стойността на едно свойство (в литературата наричан още getter) се прави в тялото на свойството, като за целта трябва да се спазва следния синтаксис:

|  |
| --- |
| get { <accessor\_body> } |

Съдържанието на блока ограден от фигурните скоби (<accessor\_body>) е подобно на съдържанието на произволен метод. В него се декларират действията, които трябва да се извършат за връщане на резултата от метода.

Методът за четене на стойността на едно свойство трябва да завършва с return или throw операция. Типът на стойността, която се връща като резултат от този метод, трябва да е същият както типa <property\_type> описан в декларацията на свойството.

**Метод за промяна на стойността на свойство (setter)**

По подобие на метода за четене на стойността на едно свойство, може да се декларира и метод за промяна (модификация) на стойността на едно свойство (в литературата наричан още setter). Той се декларира в тялото на свойството с тип на връщана стойност void и в него подадената при присвояването стойност е достъпна през неявен параметър value.

Декларацията се прави в тялото на свойството, като за целта трябва да се спазва следнияt синтаксис:

|  |
| --- |
| set { <accessor\_body> } |

Съдържанието на блока ограден от фигурните скоби (<accessor\_body>) е подобно на съдържанието, на произволен метод. В него се декларират действията, които трябва да се извършат за промяна на стойността на свойството. Този метод използва неявен параметър, наречен value, който е предоставен от С# по подразбиране и който съдържа новата стойност на свойството. Той е от същия тип, от който е свойството.

**Видове свойства**

В зависимост от особеностите им, можем да класифицираме свойствата по следния начин:

1.  Само за четене (read-only), т.е. тези свойства имат само get метод, както в примера с лицето на правоъгълник.

2.  Само за модифициране (write-only), т.е. тези свойства имат само set метод, но не и метод за четене на стойността на свойството.

3.  И най-честият случай е read-write, когато свойството има методи както за четене, така и за промяна на стойността.

**Статични класове (static classes) и статични членове на класа (static members)**

Когато един елемент на класа е деклариран с модификатор static, го наричаме статичен. В С# като статични могат да бъдат декларирани полетата, методите, свойствата, конструкторите и класовете.

Статичен член (static member) на класа нари­чаме всяко поле, свойство, метод или друг член, който има модификатор static в декларацията си. Това означава, че полета, методи и свойства маркирани като статични, принад­ле­жат на самия клас, а не на някой конкретен обект от да­де­ния клас.

Следователно, когато маркираме поле, метод или свойство като статични, можем да ги използваме, без да създаваме нито един обект от дадения клас. Единственото, от което се нуждаем е да имаме достъп (видимост) до кла­са, за да можем да извикваме статичните му методи, или да достъп­ваме статичните му полета и свойства.

|  |
| --- |
| **Статичните елементи на класа могат да се използват без да се създава обект от дадения клас.** |