*Изпитна тема № 4:* **Обектно-ориентирано програмиране**

Концепцията за типизиране на класове. Създаване на шаблонни класове и методи. Наследяване на класове. Презаписване и презареждане. Абстракция и интерфейси – връзка и разлики, употреба на абстрактни класове и интерфейси. Полиморфизъм – понятие, видове полиморфизъм. Принципи на обектно-ориентираното програмиране – примери и приложение. Създаване и/или поправка/допълване на вече съществуващи компютърни програми, описващи обекти от реалния свят със средствата на програмен език.

**Клас** (**class**) в ООП наричаме описание (спецификация) на даден клас обекти от реал­ността. Класът представлява шаблон, който описва видо­вете състояния и поведението на конкретните обекти (екземплярите), които биват създа­вани от този клас (шаблон).

**Обект** (**object**) наричаме екземпляр създаден по дефиницията (описание­то) на даден клас. Когато един обект е създаден по описанието, което един клас дефинира, казваме, че **обектът е от тип "името на този клас"**.

Например, ако имаме клас **Dog**, описващ някакви характеристики на куче от реалния свят, казваме, че обектите, които са създадени по описанието на този клас (например кученцата "Шаро" и "Рекс") са от тип класa **Dog**. Това означение е същото, както когато казваме, че низът "some string" е от класа **String**. Разликата е, че обектът от тип **Dog** е екземпляр от клас, който не е част от библио­теката с класове на .NET Framework, а е дефиниран от самите нас.

**Какво съдържа един клас?**

Всеки клас съдържа дефиниция на това какви данни трябва да се съдържат в един обект, за да се опише състоянието му. Обектът (конкретния екземпляр от този клас) съдържа самите данни. Тези данни дефинират състоянието му.

Освен състоянието, в класа също се описва и поведението на обектите. Поведението се изразява в действията, които могат да бъдат извършвани от обектите. Средството на ООП, чрез което можем да описваме поведе­нието на обектите от даден клас, е декларирането на методи в класа.

**Елементи на класа**

Сега ще изброим основните елементи на един клас, а по-късно ще разгле­да­ме подробно всеки един от тях.

Основните елементи на класовете в C# са следните:

-     **Декларация на класа (class declaration)**– това е редът, на който декларираме името на класа.

Например: public class Dog

-     **Тяло на клас**– по подобие на методите, класовете също имат част, която следва декларацията им, оградена с фигурни скоби – "**{**" и "**}**" между които се намира съдържанието на класа. Тя се нарича тяло на класа. Елементите на класа, които се описват в тялото му са изброени в следващите точки.

public class Dog

{

      // ... The body of the class comes here ...

}

-     **Конструктор (constructor)** – това е псевдометод, който се из­пол­зва за създа­ване на нови обекти. Така изглежда един конструктор:

public Dog()

{

      // ... Some code ...

}

-     **Полета (fields)** – те са променливи, декларирани в класа (някъде в лите­ратурата се срещат като **член-променливи**). В тях се пазят данни, които отразяват състоянието на обекта и са нужни за работата на методите на класа. Стойността, която се пази в полетата, отразява конкретното състояние на дадения обект, но съществуват и такива полета, наречени **статични**, които са общи за всички обекти.

// Field definition

private string name;

-     **Свойства (properties)** – така наричаме характеристиките на даден клас. Обикновено стойността на тези характеристики се пази в полета. Подобно на полетата, свойствата могат да бъдат притежа­вани само от конкретен обект или да са споделени между всички обекти от тип даден клас.

// Property definition

private string Name { get; set; }

-     **Методи (methods)** – от главата "[Методи](https://introprogramming.info/intro-csharp-book/read-online/glava9-metodi)", знаем, че методите представляват именувани блокове програмен код. Те извършват някакви действия и чрез тях реализират поведението на обектите от този клас. В методите се изпълняват алгоритмите и се обработват данните на обекта.

За момента няма да обясняваме в по-големи детайли изложения код, тъй като подробна информация ще бъде дадена при обяснението как се декларира всеки един от елементите на класа.

**Класовете** са описание (модел) на реални предмети или явления, наречени същности (entities). Например класът "Студент".

Класовете имат характеристики – в програмирането са наречени **свойства (properties)**. Например съвкупност от оценки.

Класовете имат и поведение**–**в програмирането са наречени **методи** **(methods)**. Например явяване на изпит.

Методите и свойствата могат да бъдат видими само в областта на класа, в който са декларирани и наследниците му (private/protected), или видими за всички останали класове (public).

**Обектите (objects)** са екземпляри (инстанции) на класовете. Например Иван е студент, Петър също е студент.

**Обектно-ориентирано програмиране (ООП)**

Обектно-ориентираното програмиране е наследник на процедурното (структурно) програмиране. Процедурното програмиране най-общо казано описва програмите чрез група от преизползваеми парчета код (проце­дури), които дефинират входни и изходни параметри. Процедурните прог­рами представляват съвкупност от процедури, които се извикват една друга.

Проблемът при процедурното програмиране е, че преизползваемостта на кода е трудно постижима и ограничена – само процедурите могат да се преизползват, а те трудно могат да бъдат направени общи и гъвкави. Няма лесен начин да се реализират абстрактни структури от данни, които имат различни имплементации.

Обектно-ориентираният подход залага на парадигмата, че всяка програма работи с данни, описващи същности (предмети и явления) от реалния живот. Например една счетоводна програма работи с фактури, стоки, складове, наличности, продажби и т.н.

Така се появяват обектите – те описват характеристиките (свойства) и поведението (методи) на тези същности от реалния живот.

**Основни предимства и цели на ООП** – да позволи по-бърза разра­ботка на сложен софтуер и по-лесната му поддръжка. ООП позволява по лесен начин да се преизползва кода, като залага на прости и общоприети правила (принципи). Нека ги разгледаме.

**Основни принципи на ООП**

За да бъде един програмен език обектно-ориентиран, той трябва не само да позволява работа с класове и обекти, но и трябва да дава възможност за имплементирането и използването на принципите и концепциите на ООП: наследяване, абстракция, капсулация и полиморфизъм. Сега ще разгле­даме в детайли всеки от тези основни принципи на ООП.

-     **Капсулация (Encapsulation)**

Ще се научим да скриваме ненужните детайли в нашите класове и да предоставяме прост и ясен интерфейс за работа с тях.

-     **Наследяване (Inheritance)**

Ще обясним как йерархиите от класове подобряват четимостта на кода и позволяват преизползване на функцио­налност.

-     **Абстракция (Abstraction)**

Ще се научим да виждаме един обект само от гледната точка, която ни интересува, и да игнорираме всички останали детайли.

-     **Полиморфизъм (Polymorphism)**

Ще обясним как да работим по еднакъв начин с различни обекти, които дефинират специфична имплементация на някакво абстрактно поведение.

**Наследяване (Inheritance)**

**Наследяването** е основен принцип от обектно-ориентираното програми­ране. То позволява на един клас да "наследява" (поведение и характе­ристики) от друг, по-общ клас. Например лъвът е от семейство котки. Всички котки имат четири лапи, хищници са, преследват жертвите си. Тази функцио­налност може да се напише веднъж в клас Котка и всички хищници да я преизползват – тигър, пума, рис и т.н.

**Как се дефинира наследяване в .NET?**

Наследяването в .NET става със специална структура при декларацията на класа. В .NET и други модерни езици за програмиране един клас може да наследи само един друг клас **(single inheritance)**, за разлика от C++, където се поддържа множествено наследяване **(multiple inheritance)**. Ограничението е поро­дено от това, че при наследяване на два класа с еднакъв метод е трудно да се реши кой от тях да се използва.

Класът, който наследяваме, се нарича **клас-родител** или още **базов клас (base class, super class)**.

**Модификатори на достъп на членове на класа при наследяване**

-     **protected** дефинира членове на класа, които са невидими за ползвателите на класа (тези, които го инстанцират и използват), но са видими за класовете наследници

-     **protected internal** дефинира членове на класа, които са едновременно **internal**, тоест видими за ползвателите в цялото асембли, но едновременно с това са и **protected** - невидими за ползвателите на класа (извън асемблито), но са видими за класовете наследници (дори и тези извън асемблито).

Когато се наследява един базов клас:

-     Всички негови **public** и **protected, protected internal**членове (методи, свойства и т.н.) са видими за класа наследник.

-     Всички негови **private**методи, свойства и член-променливи не са видими за класа наследник.

-     Всички негови **internal** членове са видими за класа наследник само ако базовият клас и наследникът са в едно и също асембли.

**Абстрактни класове**

Съществуват и т.нар абстрактни методи -това са методи, които съдържат само декларация, но нямат имплементация. Декларират се с ключовата дума abstract. Ако декларираме поне 1 такъв метод в рамките на даден клас, то този клас задължително трябва да съдържа в декларацията си думата abstract, тоест трябва да бъде абстрактен.  
  
**Определение**: Абстрактен клас е такъв клас, който е деклариран с ключовата дума abstract и съдържащ или не абстрактни методи.

В един абстрактен клас може да напишем както абстрактни методи, така и такива които не са абстрактни– ще ги наричаме конкретни методи.

**Важно!!!**

* Абстрактен клас **не** може да се инстанцира! Може само да се наследява.
* Ако даден клас наследи абстрактен клас, то той задължително трябва да имплементира абстрактните му методи или и той трябва да е абстрактен, ако остави дори и един от тях абстрактен.

**Защо са ни необходими абстрактните класове?**

Абстрактните класове, както подсказва името им носят известно ниво абстракция. В йерархията на наследяването те могат да обобщят определена функционалност или белези, характерни за определено ниво е нея.

**Пример:** Нека имаме клас Животно и няколко други класове – Котка, Куче, Кон. Последните три класа наследяват класа Животно, така че те имат няколко общи навици(методи), които имат всички животни, затова ги наследяват от Животно. Ако осмислите примера, ще разберете, че няма смисъл да създаваме инстанция на клас Животно, защото няма как да знаем какво е това животно. Затова спазваме следната философия в ООП: ако имаме клас, който да обобщи поведение и неспецифични, а общи белези на някои други класове, то го декларираме като абстрактен клас, а наследниците му – като конкретни класове, които **задължително**да имплементират поведението, но всеки по свой начин. Ако за Животно имаме метод(поведение) „издаване на звук“, то това поведение би било абстрактно, защото различните животни издават различни звуци. Например клас Куче ще имплементира функционалността „издаване на звук“ като „Бау-бау“, Котка – като „Мяу – мяу“ и тн. Ако създадем обект от тип Куче и извикаме метода му за издаване на звук, то ще лае, но ако създадем обект (хипотетично) от клас Животно и извикаме метода му за издаване на звук, то няма да знае какъв звук да издаде.

**Интерфейси:**

Ако имаме абстрактен клас, който съдържа само абстрактни методи, то спокойно можем да го направим на интерфейс.

**Определение** за интерфейс: Интерфейсът е подобен на абстрактния клас, но с тази разлика, че може да съдържа абстрактни методи и член променливи. В интерфейсите вече може да има както абстрактни, така и конкретни – имплементирани методи. Последните се наричат default methods – нововъведение, позволяващо да се задава функционалност по подразбиране. Променливите, декаларирани в интерфейса са само публични, статични и константни – public final static.

Деклариране на интерфейс: Става с ключовата дума **interface:**

*public interface FirstInterface{*

*/\* All the methods are public abstract by default   Note down that these methods are not having body   \*/*

*public abstract void method1();*

*public abstract void method2();*

*}*

За да може един клас да наследи интерфейса, то се използва думата **implements.** Ако един клас наследи (имплементира) интерфейс, то той задължително трябва да имплементира всички негови абстрактни методи, или да се декларира като абстрактен.

**Важно!!!**

* Интерфейс не може да имплементира друг интерфейс, но може да го наследи ако се налага. Пример: Ако един интерфейс наследява друг, то ако създадем клас, който да имплементира наследника, то той ще трябва да имплементира всички методи и на двата интерфейса, но ако се имплементира родителския интерфейс, то в дадения клас трябва да се имплементират само неговите методи.
* Интерфейс не може да се инстанцира!
* В интерфейс може да има абстрактни и default методи.
* Интерфейс може да бъде имплементиран само от клас.
* За да може клас да имплементира интерфейс, то интерфейсът трябва да е публичен, методите му също.
* Интерфейсът не може да бъде деклариран с private, protected или transient.
* По подразбиране всички методи на интерфейса са: public abstract.
* Променливите на интерфейса са public static final Например:

**Полиморфизъм**

Полиморфизмът е един от четирите принципа на обектно-ориентираното програмиране. Понятието ‘полиморфизъм’ произлиза от биологията и означава ‘много форми’. От гледна точка на програмирането се интерпретира като „един интерфейс – много методи”.

**Пример:** Различните превозни средства (леки автомобили, тирове, автобуси) имат различно устройство на двигателя. Те обаче се управляват по един и същи начин и основната им функция е да извършват движение. Всички имат: волан, скоростна кутия, мигачи, фарове… Въпреки това, движението се извършва по различен начин поради различното устройство на превозното средство. Тоест, ако трябва да обобщим: Различните обекти извършват едно и също действие по различен начин (‘един интерфейс- много форми). Именно тази тяхна характеристика наричаме ‘**полиморфизъм’**.

Практически пример за полиморфизъм е и примерът, посочен по-горе за намиране на лице на фигура- площта се намира по различен начин в зависимост от това, коя е фигурата, но в крайна сметка търсената величина остава лицето.

**Видове полиморфизъм**:

* **Статичен** полиморфизъм (compile time)
  + Ad hoc polymorphism:
    - Презаписване (overload) на членове
    - Презаписване (overload) на оператори
    - Преобразуване на типове (coercion)
* **Parametric** polymorphism
* Шаблони
* **Динамичен** полиморфизъм (run time)
* Subtype polymorphism
* Предефиниране (override) на членове
* Абстракция