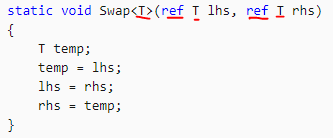
Изпитна тема № 4: Обектно-ориентирано програмиране

**План-тезис:** Компонентно тестване. Шаблонни класове и методи. Наследяване, абстракция и интерфейси. Полиморфизъм. Итератори. Компаратори. Отражение на типовете. Ламбда изрази и функции. Библиотека за обработка на колекции. Делегати. Комуникация между обекти. Изключения. Работа с потоци и файлове. Базови шаблони за дизайн.

**Компонентно тестване -** Вид софтуерно тестване, при който се прави тестване на всеки отделен компонент поотделно без да се засягат другите компоненти. Други наименования на този тип тестване биват: Единично тестване(Unit Testing), Програмно тестване(Program Testing) или Модулно тестване(Module Testing).

**Шаблонни класове -** Шаблонните класове съдържат операции, които не са специфични за определен тип данни. Шаблонните класове се използват най-много при колекции като свързани списъци, хеш-таблици, стекове, опашки, дървета и т. н. Операциите като добавяне и премахване на елементи от колекциите се правят по един и същи начин, независимо от типа данни, който се пази.

**Шаблонни методи -** Методи в C#, които се дефинират с “type” параметри. Долу на снимката е даден пример.



**Наследяване -** Едно от основните свойства на ООП. Позволява дефинирането на дъщерен клас, който преизползва(наследява), разширява или модифицира поведението на “родителски” клас. Класът, чиито параметри се наследяват, се нарича “базов клас”(base class). Класът, който наследява членовете на базовия клас се нарича “произлизащ клас”(derived class). В C# и .NET един клас може да наследи максимум един друг клас. Наследяването обаче е “транзитивно”(transitive), което ще рече, че клас “D” може да наследи клас “C”, а клас “C” да наследи клас “B” и така по-надълбоко в йерархията.

**Абстракция -** Представлява скриване на определени детайли и показване на само важна за потребителя информация. Абстракцията може да бъде постигната или с абстрактни класове, или с абстрактни интерфейси. За тази цел се използва ключовата дума “abstract” като модификатор. Абстрактният клас е ограничен клас, който не може да бъде използван за създаването на обекти(за да бъде достъпен, трябва да се наследи от друг клас), но може да съдържа както абстрактни, така и обикновени методи. Абстрактният метод може да се използва единствено в абстрактен клас и няма тяло. Тялото му се дефинира в произлизащия клас(derived class).

**Интерфейси -** Изцяло абстрактен клас, който може да съдържа само абстрактни методи и свойства(properties), с празни “тела”. По конвенция е прието имената на интерфейсите да започват с буквата “I”. По подразбиране, членовете на интерфейс са абстрактни и публични. За да се имплементира интерфейс, трябва да се използва символът “:”(като при наследяването). Тялото на интерфейсните методи се дефинира в “имплементиращия” клас. Интерфейсите не могат да се използват за създаването на обекти. При дефиниране на интерфейс не е нужно да се прави “override” на всеки от методите му.

**Полиморфизъм -** Появява се когато има много класове, които са свързани един с друг чрез наследяване. Позволява едни и същи методи да извършват различни действия. Това позволява да се изпълнява едно действие по няколко начина и преизползване на код. Модификаторът “virtual” позволява базов метод да бъде преизползван, но произлизащите от него методи трябва да бъдат с модификатори “override”.

**Итератори -** Използва се за преминаване през колекции(списъци и масиви). Итериращ метод или “get” достъпвател(accessor) прави специфична итерация върху дадена колекция. Итераторът използва “yield return” обявление, за да върне всеки елемент по веднъж. Когато такова обявление бъде достигнато, моментната локация в кода се запомня. Изпълнението се рестартира от тази локация следващия път, когато итериращата функция бъде извикана. Итератори биват използвани при употребата на “foreach” цикли или LINQ заявки.

**Компаратори -** В C# има два класа, които служат за сравняването на данни - “Comparer<T>” и “IComparer<T>”. Единият е интерфейс, съдържащ декларация на функция за сравняване на обекти по стойности по подразбиране(обикновено стрингове), а другият е клас, който дефинира произлизаща функция за сравняване. Те се достъпват при използването на оператори като “==” или извикване на метода “.Equals” или “.Compare”.

**Отражение на типовете(Reflection) -** Предлага обекти, които описват “сглобки”(assemblies), модули и типове. Може да се използва за динамично създаване на инстанция на тип и свързване на типа към съществуващ обект или за вземане на тип от съществуващ обект и извикване на неговите методи или достъпване на неговите полета и свойства. Ако са използвани атрибути в кода, отражението на типовете позволява те да бъдат достъпвани.

**Ламбда изрази и функции -** Набор от функции, които са дефинирани на един ред, без тяло. Делят се на ламбда изрази(имат израз за тяло) и ламбда обявления(имат набор от обявления в тялото си). Операторът за ламбда декларации е “=>” и той разделя списъка с параметри на ламбда функцията/израза от нейното тяло. Параметрите се указват от лявата страна на оператора, а израза или обявлението - от дясната. Всеки ламбда израз може да бъде превърнат в делегатен тип, а това зависи от типовете на параметрите и връщаната стойност.

**Библиотека за обработка на колекции -** Библиотеката LINQ(Language-Integrated Query) може да бъде използвана за достъпване на колекции. LINQ заявките предоставят функционалности като филтриране, подреждане и групиране.

**Делегати -** Тип, който представлява референции към методи с определен набор от параметри и определен връщан тип. Когато делегат бива инстанциран, неговата инстанция може да бъде свързана с всеки метод, който има сходни параметри и връщан тип. Методът може да бъде извикван през инстанцията на делегата. Делегатите се използват за подаване на методи като аргументи на други методи.

**Комуникация между обекти -** В C# съществуват два класа, които отговарят за комуникацията между обекти - CommunicationObject и ICommunicationObject. CommunicationObject класа имплементира ICommunicationObject интерфейса за всички комуникационни обекти в Windows Communication Foundation(WCF). Имплементациите на методите отговарят за валидацията на параметрите, активират събития, които известяват относно появата на промени в състоянията, предоставят средствата към свързаното допълнително обработване с тези промени в състоянието и уверяват, че тези промени са координирани.

**Изключения -** Произлизат от класа System.Exception и трябва да бъдат хващани с try-catch комбинации от команди. Ако не бъде намерен подходящ сценарий за съответния вид изключение, програмата спира да работи и се връща съобщение за грешка. Изключенията се използват за връщане на подробна информация при възникване на грешка в дадена програма.

**Работа с потоци и файлове -** Входът и изходът на файлове и потоци се отнася към пренасянето на данни към или от дадено място за съхранение. В .NET библиотеките на System.IO съдържат типове, които позволяват четенето и писането, както синхронно, така и асинхронно на потоци от данни и файлове. Тези библиотеки също така съдържат типове, които извършват компресия и декомпресия на файлове и типове, които позволяват комуникация през работни потоци и серийни портове. Файловете представляват подредена и наименована колекция от байтове, която има постоянно съхранение. Потоците са поредици от байтове, от които може да се чете и да се пише на различни места за съхранение(дискове или памет). Освен файлови потоци съществуват мрежови, съхранителни(memory) и работнолинейни(pipe) потоци.

**Базови шаблони за дизайн -** Използват се, за да се дефинират основните стъпки на даден алгоритъм и да се разреши имплементацията на индивидуалните стъпки да бъде променяна. Този тип модели за дизайн е близък до стратегическия модел за дизайн. Главната разлика е възможността да се сменят части от алгоритъма вместо целия алгоритъм. При базовите шаблони за дизайн абстрактен клас дефинира шаблонния метод и този метод включва стъпките, които се имплементират от класовете-наследници.

Условие на приложната задача:

<https://drive.google.com/file/d/18fBvuLk2fi6zaIKmEOEQQ0DtwNDAtLGn/view?usp=sharing>

Решение на приложната задача:

<https://github.com/VKarazhekov/izpitni_temi_Vasil_Karazhekov/tree/master/Izpitna_tema_4_OOP>