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

1.

1.1. Дефинира: клас, конструктор, поле, свойство. (2 т.)

- *Клас* – Дефинира характеристиките на даден обект и неговото поведение. Той е план или шаблон, чрез който се описва природата на някакъв обект.
- *Конструктор* – Специален метод, който се извиква при създаването на обект от даден клас и извършва инициализация на данните му.
- *Поле* – Променливи, декларирани в класа (член-променливи), които отразяват състоянието на обекта и са нужни за работата на методите на класа.
- *Свойство* - 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.

1.2. Описва създаване на обекти от клас. (2 т.)

Създаваме класа Cat.

```
public class Cat
{
    public string Name { get; set; }

    public string Color { get; set; }
}
```

Създаване на обект от клас Cat:

```
Cat someCat = new Cat();
```

Създаването на обекти става чрез оператора new, който заделя нужната памет на хийпа. Новосъздадения обект обикновено се присвоява на променлива от тип съвпадащ с класа на обекта.

На променливата someCat присвояваме новосъздадена инстация на класа Cat.

1.3. Различава понятията конструктор, метод и свойства. (4 т.)

Методите представляват именувани блокове програмен код. Те извършват действия и чрез тях реализират поведението на обектите на този клас.

Конструкторът е специален метод, който се използва за създаване на обекти (инициализирането на полетата/свойствата на обекта).

Свойствата в общия случай са специални методи за косвен достъп до полетата на даден клас.

2.

2.1 Дефинира понятията функции/методи, тип и стойност на връщане, параметри и посочва видове параметри на функция/метод. (2 т.)

Методът е именувано парче код, което решава даден проблем и които могат да бъдат извикани толкова пъти, колкото имаме нужда. Методът може да приема параметри и да връща стойност. В различните езици могат да се срещат и като функции. Декларация на метод:
<модификатор за достъп><тип на върната стойност><име на метода(идентификатор)>(<списък от формални параметри(аргументи))>

Връщаната стойност може да бъде както int , double , string и т.н., така и void . Ако типът е void, то това означава, че методът не връща резултат, а само изпълнява дадена операция или манипулира данни без да връща краен резултат.

Параметрите биват три вида: входни, изходни, входно-изходни.

Таблица, показваща различията между трите вида параметри			
Вид	входни	изходни	входно-изходни
Предават се	по стойност	по адрес	по адрес
Фактически параметър е	израз от типа на формалния	променлива от типа на формалния	инициализирана променлива от типа на формалния
Методът работи с	копие на фактически параметър	оригинала на фактическия параметър	оригинала на фактическия параметър
Служебна дума	няма	out	ref
Примери	(int a, float x)	(out int s, out int pr)	(ref int s, ref int pr)

2.2 Демонстрира дефинирането и употребата на функции/методи. (6 т.)

Метод, който принтира четните числа от 0 до подаденото като параметър число.

```
static public void PrintAllEvenNumbersTo(int range)
{
   for (int i = 2; i < range; i++)
   {
      if (i % 2 == 0)
      {
         Console.WriteLine(i);
      }
   }
}
```

```
static void Main(string[] args)
{
        int range = int.Parse(Console.ReadLine());
        PrintAllEvenNumbersTo(range);
}
```

3.

3.1. Прави заключения и изводи за употребата на ключовата дума this. (6 т.)

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 от създателите на езика, с която да достъпваме елементите (полета, методи, конструктори) на собствения ни клас:

```
this.myField; // access a field in the class
this.DoMyMethod(); // access a method in the class
this(3, 4); // access a constructor with two int parameters
```

3.2. Дава пример за ситуации, които показват необходимостта от употребата на ключовата дума this. (6 т.)

Когато името на поле от класа съвпада с името на параметър на конструктора използваме this, за да модифицираме полето. Употребата на оператор this може да бъде избегната с прилагането на конвенциите за именуване:

- Използване на долна черта пред името на полето. Пример _name
- Използване на m_ пред името на полето, за да се покаже, че полето е член (мембър) на класа. Пример m_name

```
public class Dog
 {
    private string name;

    public Dog(string name)
    {
       this.name = name;
    }
 }
```

4.

4.1. Обяснява енкапсулирането на данни в класовете. (4 т.)

Енкапсулация – 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 с възможно най-ограничено ниво на видимост (private/protected) и декларираме достъпът на тези полета да може да се осъществява единствено чрез специални методи за достъп.

4.2. Дава пример за употребата на методите за достъп и промяна на енкапсулираните данни. (4 т.)

```
using System;

class Point
{
	private double m_x;
	private double m_y;

	public Point(int x, int y)
	{
		m_x = x;
		m_y = y;
	}

	public double X
	{
		get { return m_x; }
		set { m_x = value; }
	}

	public double Y
	{
		get { return m_y; }
		set { m_y = value; }
	}
}
```

```
using System;

class PointTest
{
	static void Main()
	{
		Point myPoint = new Point(2, 3);

		double myPointXCoordinate = myPoint.X; // Достъпване на
свойство чрез get
		double myPointYCoordinate = myPoint.Y; // Достъпване на
свойство чрез get

		myPoint.X = 5; // Промяна на свойство чрез set
```

```
            myPoint.Y = 6; // Промяна на свойство чрез set
        }
    }
```

5.

5.1. Дефинира и описва статичен клас и статични членове на класа. (2 т.)

Статичен клас - декларира се с помощта на ключова дума static. Статичният клас може да съдържа само статични членове на данни, статични методи и статичен конструктор. Не е позволено да се създават обекти от static клас. Статичните класове са запечатани, което означава, че не може да се наследи статичен клас от друг клас. Използват се при дизайн патерн от вида Singelton и когато сме сигурни, че дадения клас няма да бъде да наследяван и че ще съдържа методи, с които да се обработват външни за класа данни. Пример за такъв клас е Math.

Статични променливи - декларират се с ключовата дума static. Когато една променлива е статична се създава едно нейно копие, което се споделя между всички обекти на ниво клас. Статичните променливи се достъпват чрез името на класа, не изискват обект.

Статичен метод - статичен метод се декларира с ключовата дума static. Статичните методи се достъпват чрез името на класа. Един статичен метод може да достъпва статични (директно достъпвани без името на класа) полета и не-статични полета (достъпвани чрез обекти).

Статичен конструктор - декларира се също с ключовата дума static. Той се извиква само веднъж при създаването на първата референция към статичен член на класа. Статичният конструктор инициализира статичните полета на класа и се извиква само веднъж.

5.2. Дава пример за употребата на статични членове в клас. (4 т.)

Пример: Програма със статичен клас

```
using System;
static class Tutorial
{
   public static string Topic = "Static class";
}
public class Program
{
   static public void Main()
   {
      Console.WriteLine("Topic name is : {0} ", Tutorial.Topic);
   }
}
```

Пример: Програма със статична променлива

```
using System;
namespace tema3csh
{
class Vehicle
   {
      public static string Model_color = "Black";
      public override string ToString()
      {
         return (" " + Vehicle.Model_color);
      }
   }
   public class Program
   {
      static public void Main()
      {
         Console.WriteLine("Color of the all models is  : {0} ", Vehicle.Model_color);

         Vehicle car = new Vehicle();
         Console.WriteLine("Color of the car is  : {0} ",Vehicle.Model_color);

         Vehicle motorcycle = new Vehicle();
         Console.WriteLine("Color of the motorcycle is :"+motorcycle.ToString());
      }
   }
}
```

Пример: Програма с използване на статичен метод

```
using System;
namespace tema3csh1
{
   class Nparks
   {
       static public int t = 11;
      public static void Total()
      {
         Console.WriteLine("Total number of national parks" +
                        " present in Bulgaria is :{0}", t);
      }
   }
   public class Program
```

```
    {
        static public void Main()
        {
            Nparks.total();
        }
    }
}
```

Изход:
Total number of national parks present in Bulgaria is: 11

Пример: Програма с използване на статичен конструктор

```
using System;
namespace tema3csh2
{
    class G1
    {
        static G1()
        {
            Console.WriteLine("Example of Static Constructor");
        }

        public G1(int j)
        {
            Console.WriteLine("Instance Constructor " + j);
        }

        public string g1_detail(string name, string branch)
        {
            return "Name: " + name + " Branch: " + branch;
        }

        public static void Main()
        {
             G1 obj = new G1(1);

            Console.WriteLine(obj.g1_detail("Sunil", "CSE"));

            G1 ob = new G1(2);

            Console.WriteLine(ob.g1_detail("Sweta", "ECE"));
```

```
        }
      }
    }
```

6.

6.1. Дефинира по-сложни класове. (2 т.)

Под по-сложни класове може да се разбира:

- Класове, които да съдържат по-сложна логика на опериране като класът Math и Array.
- Класове, които да съдържат много на брой методи, полета, свойства, конструктури. Както и да са свързани в наследствена верига, да реализират шаблонни операции и да съдържат структури от данни, реализиращи операции свързани с данните на класа.

6.2. Различава модификатори за достъп. (4 т.)

Ниво на достъп public - 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#.

Ниво на достъп private - 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.

Ниво на достъп protected – Модификаторът protected се използва, както модификатора private, но с разликата, че при наследствена верига, наследниците имат достъп до членовете декларирани като protected, но нямат достъп до тези, които са декларирани като private.

Ниво на достъп internal - 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.

6.3. Разработва по-сложни класове с правилна енкапсулация на членовете на класа. (6 т.)

```
public class Person
{
   private string m_firstName;
   private string m_lastName;
   private int m_age;
   private decimal m_salary;

   public Person(string firstName, string lastName, int age, decimal salary)
   {
      m_firstName = firstName;
      m_lastName = lastName;
      m_age = age;
      m_salary = salary;
   }

         public string FirstName
         {
                get => m_firstName;
                private set => m_firstName = value;
         }

   public string LastName
        {
                get => m_lastName;
                private set => m_lastName = value;
        }

   public int Age
        {
                get => m_age;
                private set m_age = value;
        }

   public decimal Salary
        {
                get => m_salary;
                private set => m_salary = value;
        }
}
```