class Knight

{

    public string Name

    public int Strength

    public int Health;

public static int KnightCount;

//konstruktor s defaultním nastavením

    public Knight(string name)

    {

        this.Name = name;

        this.Strength = 20;

        this.Health = 100;

    }

 //\*konstruktor bez defaultního nastavení

    public Knight(string name, int strength, int health)

    {

        this.Name = name;

        this.Strength = strength;

        this.Health = health;

System.Console.WriteLine("Inside of the big constructor.");

    }

//\*\*výše uvedené 2 konstruktory mohu nahradit (zjednodušit) tímto jedním s povinným jménem

    public Knight(string name, int strength = 20, int health = 100)

    //public Knight(string name = "Null", int strength=20, int health=100) by vzal i zadání bez jména

    {

        this.Name = name;

        this.Strength = strength;

        this.Health = health;

Knight.KnightCount += 1;

    }

/nebo můžeme volat konstruktory mezi sebou

    public Knight(string name) :this(name, 20,100) //tento small konstruktor volá big konstruktor bez defaultního nastavení\* nebo s defaultním nastavení\*\* (v pořadí big small)

    {System.Console.WriteLine("Inside of the small constructor.");}

public Knight(string nam = "Null") :this(name, 20,100); // umožní vytvoření (inicializaci) instance Knighta i bez jména

//nemohu mít následující dva konstruktory současně, protože neumí rozlišit 2 int (stejný datový typ) když zadám např. Knight jan = new Knight(10);

    public Knight(int strength) : this("Null", strength, 100)

    { }

    public Knight(int health) : this("Null", 20, health)

    { }

private int Health; //změním Health na private

public void AddHealth(int health) //můj setter - nutný když změním field Health na private (a nevyužiji {set;}), metoda může měnit i private fieldy

    {

        if (health < 10)

        {

            this.Health += health; }}

public void DefendYourself(Knight attacker) //můj setter

    {

        this.Health -= attacker.Strength;

    }

public int GetHealth() //můj getter pro private field NEBO to lze umožnit pomocí {get;}

    {

        return this.Health;

    }

//místo AddHealth(int health) a GetHealth() chci použít {get; set;}:

private int \_health; //tuto private hodnotu potřebuji pro custom ověření - viz public int Health

public static int KnightCount { get; private set; } //příklad fungování static

public string Name { get; } //if nenastavím public bude defaultně private, dám pouze {get;} abych nemohla změnit

public int Strength { get; } //nemám zde {set;} tzn. že mohu nastavit jen v konstruktoru => zabráním nežádoucí změně

//public int Health { get; set; } //takto by mi fungovalo nastavení geralt.Health += 5; ale tomu chci zabránit

public int Health

    {

        get

        {

            return \_health; //bude nahrazeno klíčovým slovem field return field;

        }

        private set //(int value)

        {

            if (value < 0)

            {

                \_health = 0; //field = 0;

            }

            else

            {

                \_health = value; //field = value;

            }

        }

    }

//analogie s metodou (if bych místo (int health) si to pojmenovala (int value)):

    public void AddHealth(int value) //můj setter - nutný když změním Health na private (a nevyužiji {set;})

    {

        if (\_health < 10)

        {

            this.\_health += value;

        }

    }

//Console.WriteLine(geralt.Health); mi bude fungovat díky {get{}}

public string Nickname

    {

        get

        {

            return "Some guy"; //toto už nezměním - nemůžu si uložit jinou hodnotu

        }

    }

    public bool IsAlive //property, volám pomocí Console.WriteLine(geralt.IsAlive);

    {

        get

        {

            return this.\_health > 0;

        }

    }

    public bool IsAliveMethod() //metoda - udělá to stejné, volám pomocí Console.WriteLine(geralt.IsAlive());

    {

        return this.\_health > 0;

    }