КОНЦЕПТ ООП

Основни принципи објектно орјентисаног програмирања (ООП) су:

- **Енкапсулација**
- **Наслеђивање**
- Полиморфизам
- Апстракција

Енкапсулација или учауривање по којем је информација у класи заштићена од директог приступа и једини начин да се примени је кроз дефинисане методе. Енкапсулација се остварује поделом чланова класе на јавне и приватне. Поља тј. атрибути су најчешће приватна, док су методе већином јавне.

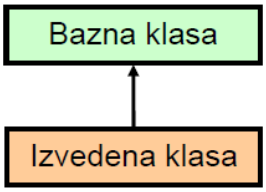
Наслеђивање је најважнији концепт ООП. По њему се из једне дефинисане класе може извести нова. Она наслеђује све чланове класе и уводи своја специфична поља и методе. Изведена класа је поткласа а базна тј. основна класа је надкласа тј. родитељска класа.

Полиморфизам потиче од грчких речи поли (много), морфе (облика). Полиморфизам је особина ООП по којем се метода понаша различито у различитим објектима. Дакле, могућност да различити објекти одговоре на исту поруку на различите начине.

Апстракција је поједностављање сложене реалности моделовањем класа које одговрају проблему. То је констролисано изостављање карактеристика реалног објекта које нису битне за решавање одређеног проблема.

НАСЛЕЂИВАЊЕ

Концепт наслеђивања омогућава да се на основу постојеће класе (базне класе) изведене нова класа (изведена класа).



Синтакса наслеђивања:

```
class BaznaKlasa {

    //clanovi bazne klase
}

class IzvedenaKlasa : BaznaKlasa {

    //clanovi izvedene klase
}
```

Правила која важе за изведену класу:

- Изведена класа наслеђује све чланове базне класе осим конструктора и деструктора. Обрнуто не важи, дакле основна класа не садржи чланове изведене класе.
- У изведеној класи можемо дефинисати нове чланове класе.
- Приватни чланови класе, иако наслеђени, су дотупни искључиво члановима базне класе.
- Чланови базне класе са модификатором **protected** су доступни унутар базне и изведене класе.
- Ако је базна класа приватна, изведена класа не може бити јавна.
- При креирању објекта изведене класе, када смо радили конструкторе, рекли смо да се прво позива (подразумевани) конструктор базне класе, па конструктор изведене класе. Исто важи и за деструкторе.

Кључна реч base

Кључну реч **base** користимо:

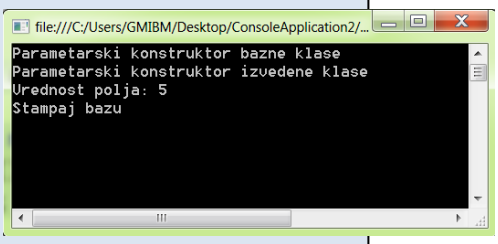
1. када позивамо конструктор основне класе из изведене класе.
2. када позивамо метод и поље основне класе из изведене класе.

1. Задатак: Дефинисане су две класе: **BaznaKlasa** и **IzvedenaKlasa**.

Базна класа састоји се од подразумеваног и параметарског конструктора, атрибута (**i**) и методе **StampajBazu**().
Изведена класа састоји се од подразумеваног и параметарког конструктора, и методе **StampajIzvedenu**()

Илустровати како се преко кључне речи **base** позива конструктор, метода и поље базне (родитељске) класе.

```
using System;

class BaznaKlasa {
    protected int i = 5;

    public BaznaKlasa() {
        Console.WriteLine("Konstruktor bazne klase");
    }

    public BaznaKlasa(int n) {
        Console.WriteLine("Parametarski konstruktor bazne klase");
    }

    public void StampajBazu() {
        Console.WriteLine("Stampaj bazu");
    }
}
class IzvedenaKlasa : BaznaKlasa {
    public IzvedenaKlasa() {
        Console.WriteLine("Konstruktor izvedene klase");
    }

    public IzvedenaKlasa(int a, int b): base(a)  //probati base()
    {
        Console.WriteLine("Parametarski konstruktor izvedene klase");
    }

    public void StampajIzvedenu(){
        //pozivanje polja bazne klase
        Console.WriteLine("Vrednost polja: {0}", base.i);
        //pozivanje metode bazne klase
        base.StampajBazu();
    }
}

class Program
{
    static void Main(string[] args)
    {
            IzvedenaKlasa p = new IzvedenaKlasa(5,3);
            p.StampajIzvedenu();
            Console.ReadKey();
    }
}
```



`IzvedenaKlasa p = new IzvedenaKlasa(5,3);`

Позива се параметарски конструктор базне класе и изведене класе.

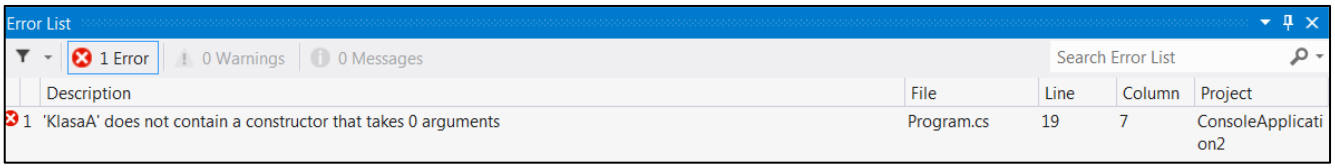
`p.StampajIzvedenu();`

Позива се метода изведене класе StampajIzvedenu(). Преко ове методе позива се атрибут родитељске класе (base.i) и метод родитељске класе (base.StampajBazu();)

2. Задатак : Анализирати наведени пример и проценити његову тачност.

```
using System;

class Program
{
    static void Main(string[] args)
    {
        KlasaB b = new KlasaB();
        b.Print();
        Console.ReadKey();
    }
}
class KlasaA {

    string s;
    public KlasaA(string s) { this.s = s; }
    public void Print() { Console.WriteLine(this.s); }
}

class KlasaB : KlasaA {
}
```

1. Задатак приказује грешку: класа **KlasaA** нема подразумевани конструктор. Експлицитним додавањем подразумеваног конструктора **KlasaA** класи програм неће имати грешку.



2. Други начин за решавање проблема био би уклањање параметарског конструктора **KlaseA** класе.

3. Задатак : Анализирати наведени пример и проценити његову тачност.

```
using System;

class Program
{
    static void Main(string[] args)
    {
        A a = new A();
        a.Print();
        Console.ReadKey();
    }
}
class A {

    string s;
    public A(string s) { this.s = s; }
    public void Print() { }
}
```

Овај пример није везан за наслеђивање алије проблем сличан као у претходном задатку. Проблем се може решити на два ничина:

- Експлицитним додавањем подразумеваног конструктора или
- Креирањем објекта при чему се позива параметарски конструктор нпр. A a = new A(“nekistring”);

Кључна реч sealed (запечаћена)

Кључну реч **sealed** онемогућава наслеђивање. Дакле, **sealed класа** не може се наслеђивати.

АНАЛИЗИРАТИ ПРИМЕРЕ И ЗАОКРУЖИТИ ТАЧАН ОДГОВОР:

161.	Процес скривања неких података о објекту од корисника и пружања само неопходних података, зове се: 1. Полиморфизам 2. Наслеђивање 3. Енкапсулација 4. Апстракција	**1**

174.	У програмском језику C# користи се службена реч ***base***. Који од наредних исказа који дефинишу дату службену реч, **<u>није</u>** тачан. 1. Службена реч ***base*** може послужити за позивање конструктора родитељске класе. 2. Службена реч ***base*** може послужити за позивање приватних метода родитељске класе којима се другачије не може приступити. 3. Службена реч ***base*** може послужити за позивање заклоњеног метода родитељске класе. 4. Службена реч ***base*** може послужити за позивање заклоњеног поља родитељске класе.	**1**

Очигледно је да су одговори под 1, 3 и 4 тачни. А одговор под 2 није тачан.

172.	Једна од основних особина објектно оријентисаног језика је наслеђивање. Дате су насловне линије дефиниције класе. Заокружити број испред дефиниције класе која се **<u>не</u>** може наследити: 1. class A { } 2. class A { private A(){ } } 3. sealed class A { } 4. class A { protected A(){ } }	**1**

218.	У програмском језику C# дата је декларација две класе: ***KlasaA*** и ***KlasaB*** која наслеђује класу ***KlasaA***. Анализирати дате класе и проценити који од понуђених исказа су тачни.
	
	`namespace TestPrimer {`
	`classProgram {`
	`staticvoid Main(string[] args) {`
	`KlasaB b = newKlasaB();`
	`b.Print();`
	`}`
	`}`
	`classKlasaA {`
	`string s;`
	`public KlasaA(string s) { this.s = s; }`
	`publicvoid Print() { Console.WriteLine(this.s); }`
	`}`
	`classKlasaB :KlasaA{ }`
	`}`
	1. Програм има грешку, јер ***KlasaB*** нема подразумевани конструктор ***KlasaB***().
	2. Програм има грешку јер ***KlasaB*** има подразумевани конструктор, док родитељска ***KlasaA*** нема такав конструктор. Програм би радио без грешке уколико би се уклонио конструктор са параметрима из ***KlasaA***.
	3. Програм има грешку која се може отклонити уколико би се у ***KlasaA*** експлицитно додао конструктор без параметара ***KlasaA***().
	4. Програм нема грешку, извршава се, али се на конзоли ништа не исписује јер је поље ***s*** добило подразумевану вредност ***String.Empty***
		2

220. Дат је код програма у програмском језику C# и састоји се од две класе у једној датотеци. Анализирати дати код и проценити његову тачност:

```
namespace TestPrimer{
classProgram{
staticvoid Main(string[] args){
A a = newA();
            a.Print();
        }
    }
classA{
string s;
public A(string s) { this.s = s; }
publicvoidPrint(){
Console.WriteLine(s);
        }
    }
}
```

1. Програм има грешку, јер класа A није јавна класа.
2. Програм има грешку, јер класа A нема подразумевани конструктор.
3. Програм нема грешака и нормално се извршава ништа не приказујући на екрану.
4. Програм има грешку која се може исправити уколико се у четвртом реду наредба **A a = new A();** замени наредбом **A a = new A("poruka");**.

3

207. Заокружити бројеве испред наведених чланова класе који се ни под којим условима **не** наслеђују са родитељске класе на изведену класу:

1. Readonly својства
2. Заштићени чланови класа
3. Својства (property)
4. Приватни чланови класа
5. Конструктор класе

1

Када смо навели правила која важе за наслеђивање, навели смо да се из родитељске на изведену класу **не наслеђују** само конструктори и деструктори. За приватне чланове родитељске класе, правилна формилација би била да се наслеђују али нису доступни изведеној класи. Практично као да нису ни наслеђени у изведену класу. Тако да се за ово питање тражи одговор под 4 и 5.

Када кажемо "заштићени чланови класе", мислимо на атрибуте – они се наслеђују као и својства и Readonly својства.

Када смо раније изучавали "Класе", проучавали смо и "модификаторе приступа". Следеће питање односи се на ту област.

235. Са леве стране су наведене области видљивости појединих елемената класе, а са десне стране класификатори приступа којима се врши контрола области видљивости. На линију испред класификатора приступа унети редни број под којим је наведена одговарајућа област видљивости:

1. private	_____	видљив унутар класе у којој је дефинисан, као и унутар изведених класа
2. public	_____	видљив само унутар класе у којој је дефинисан
3. protected	_____	видљив унутар пројекта у коме је дефинисан
4. internal	_____	видљив и ван своје класе у којој је дефинисан

2