ТЕОРИЈА: ПРОГРАМСКИ ЈЕЗИК C#

149. У програмском језику C# класа може да садржи статичка и не-статичка (инстанцна) поља. Дате су изјаве које се односе на статичка поља класе и међу њих је уметнута једна изјава која се односи на не-статичка (инстанцна) поља класе.
Заокружити број испред изјаве која се односи на не-статичка поља класе:

1. Поље које се може користити без конструисања и једног објекта те класе
2. Поље које има исту вредност за све креиране објекте неке класе
3. Поље чија се вредност може разликовати за сваки појединачни објекат неке класе
4. Поље које се може користити унутар статичких метода класе, као и унутар метода инстанце

ОДГОВОР: 3.
По C# стандарду, одговори 1, 2 и 4 су нетачни. Стандард у тим случајевима захтева статичка поља. Једино одговор 3 може да се примени на нестатичка поља.

150. Дати су типови променљивих у програмском језику C#. Како се назива променљива која је дефинисана унутар неког метода:

1. Глобална променљива
2. Статичка променљива
3. Блоковска променљива
4. Локална променљива

ОДГОВОР: 4.
Променљиве које су дефинисане унутар неког метода називају се локалне променљиве. Променљиве ван методе се називају глоабалне променљиве.

151. Дате су програмске јединице у програмском језику C#. Одредити која програмска јединица представља шаблон за конструисање објекта истог типа:

1. Класа
2. Именски простор
3. Нестатичка метода
4. Променљива
5. Статичка метода

ОДГОВОР: 1.
Класа служи за констрисање објекта. То значи да је једна класа заправо шаблон на основу кога се конструишу објекти. За објекат констриусан на основу дефиниције неке класе се каже да припада тој класи или да је **инстанца** те класе.

152. Одредити какви могу бити чланови класе (поља и методе) у програмском језику C#:

1. Локални и глобални
2. Процедурални и непроцедурални
3. Статички (класни) и нестатички (објектни)
4. Спољашњи и унутрашњи

ОДГОВОР: 3.
По дефиницији чланови класе (поља и методе) могу бити статички и нестатички.

153. Дата је наредба кода у програмском језику C# која представља дефиницију низа niz. Која је од доле наведених дефиниција низа коректна:

1. int[] niz = new int[20];
2. int[] niz = new int(20);
3. int niz = new int[20];
4. int niz() = new int[20];

ОДГОВОР: 1.
По дефиницији **int[] niz = new int[20];** Други одговор није због мале заграде, трећи одговор није због недостатка заграда а четврти одговор није јер се користе мале заграде.

154. Дат је део кода који је написан на C# програмском језику. Одредити шта ће бити на излазу.

```
class Test
{
        public static void Main(string[] args){
        Test p = new Test();
        p.start();
        }
        void start() {
                bool b1 = false;
                bool b2 = fix(b1);
                Console.WriteLine(b1 + " " + b2);
        }
        bool fix(bool b1){
                b1 = true;
                return b1;
        }
}
```

1. true true
2. false true
3. true false
4. false false

ОДГОВОР: 2.

Променљива **b1** се директно исписује дакле **false.** Затим се позива функција **fix,** која чија вредност је вредност локалне променљиве b1, дакле **true**.

155. Дати код програма у програмском језику C# састоји се од две класе у једној датотеци. Анализирати дати код и проценити његову тачност.

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args) {
                        Klasa a = new Klasa();
                        a.n++;
                }
        }
        class Klasa {
                private int n;
                private Klasa() { }
        }
}
```

1. Програм има грешку јер класа ***Klasa*** има приватни конструктор и приватно поље ***n***.
2. Програм има грешку јер класа ***Klasa*** има празан подразумевани конструктор.
3. Програм има грешку јер променљива ***n*** није иницијализована.
4. Програм нема грешака и нормално се извршава

ОДГОВОР: 1.

Уколико за конструктор не ставимо да је јавна класа тј. **public,** не можемо да је користимо изван класе у којој је.
Пошто је конструктор типа **private** (налази се у класи **Klasa**), као и **поље n**, не може се користити у класи **Program**. Када би ставили **private** уместо **public**, одговор би био под 4. што није у овом случају. Дакле, све што је приватно је видљиво само у тој класи, не изван ње, као што овде покушава да се користи.

156. У програмском језику C# декларисане су статичке променљиве логичког, нумеричког и класног типа.

```
namespace Test{
        classProgram
        {
                public static bool x;
                public static int y;
```

```
        public static Random r;
        static void Main(string[] args) { … }
        }
}
```

Одредити које аутоматске почетне (default) вредности декларисана поља имају унутар методе Main. Понуђени одговори су наведени у наглашеном редоследу:

	X	y	r
1.	true	1	null
2.	false	0	null
3.	true	0	null
4.	false	1	null
5.	false	0	void

ОДГОВОР: 2.

По дефиницији, погледати следећи линк (https://msdn.microsoft.com/en-us/library/83fhsxwc.aspx) за следеће променљиве су подразумеване вредности: Логичке променљиве (**bool = false**), нумеричке променљиве (**int = 0**) а класног типа (**Random r = null**).

157. Заокружити број испред исправно написане наредбе кода у програмском језику C#, којом се копија низа x додељује низу y:

1. Array.Copy(x, y, x.Length);
2. Array.Copy(x, y);
3. y = Array.Copy(x, x.Length);
4. y =x.CopyTo();

ОДГОВОР: 1.

Општи облик Методе **Array.Copy** је:

Array.Copy (извор за копирање, дестинација, број чланова);

На пример ако копирамо низ **x** у низ **y**, при чему копирамо све чланове из низа **x** написаћемо:

Array.Copy (x, y, x. Length);

158. У програмском језику C# дефинисана је метода са аргументом низовног типа. Одредити шта се тачно преноси том методом:

1. Копија датог низа
2. Копија првог елемента датог низа
3. Референца на дати низ
4. Дужина датог низа

ОДГОВОР: 3.

Низови су референтни типови, па се тако и преносе.

159. Дата је наредба кодау програмском језику C#, која представља позив методе класе Array. Одредити којим од датих позива се сортира низ brojevi типа int[]:

1. Array(brojevi)
2. brojevi.Sort()
3. Array.SortArray(brojevi)
4. Array.Sort(brojevi)

ОДГОВОР: 4.

Array.Sort() метода врши сортирање задатог низа. Например, ако је низ:
int [] niz = {4, 7, 2, 0}; после извршавање методе tj. **Array.Sort(niz);** вредности распоред елемената низа биће: **0, 2, 4, 7.**

```
int [] brojevi = new int[]{1,2,3};
Array(brojevi);
brojevi.Sort();
Array.SortArray(brojevi);
Array.Sort(brojevi);
```

160. У програмском језику C# дата је декларација низа:

int к;

int[] brojevi = {5, 12, 37, 7, 27, 33, 36};

На основу дате декларације одредити шта је резултат позива *k=Arrays.BinarySearch(brojevi, 37);*

1. k=0, јер метод BinarySearch прво изврши сортирање низа у опадајућем редоследу, па онда тражи задату вредност
2. метод binarySearch баца изузетак увек када је низ неуређен и програм „пуца“
3. k=2, јер се тражени елеменат налази на позицији 2
4. k добија неочекивану вредност јер низ мора бити сортиран у растућем поретку пре позива методе BinarySearch
5. k=6, јер метод BinarySearch прво изврши сортирање низа у растућем редоследу, па онда тражи задату вредност

ОДГОВОР: 4.

Метод **BinarySearch** захтева да низ буде сортиран у растућем редоследу. Ако није, повратна вредност је бесмислена. На пример ако би смо, сортирали низ (поставили да int[] brojevi = { 5, 7, 12, 27, 33, 36, 37}; онда би к било 6.

161. Процес скривања неких података о објекту од корисника и пружања само неопходних података, зове се:

1. Полиморфизам
2. Наслеђивање
3. Енкапсулација
4. Апстракција

ОДГОВОР: 3.

Полиморфизам се заснива на идеји да метода која је декларисана у основној класи може да се имплементира на више различитих начина у различитим изведеним класама.

Наслеђивање представља један од концепата Објектно Оријентисаног Програмирања који омогућава да се на основу постојеће класе изведе нова класа.

Енкапсулација или учауривање налаже да сва објектна поља буду сакривена и дефинисана са модификатором private. Један од предности овог приступа је то што су на тај начин сва поља класе заштићени унутар “чауре” класе - сакривени и могу се мењати само на контролисан начин.

Апстрактна класа је класа која се може наследити, али се не може инстацирати.

162. Која су поља заједничка и јединствена за све креиране објекте неке класе дефинисане у објектно оријентисаном програмском језику C#:

1. Јавна
2. Приватна
3. Објектна
4. Инстанцна
5. Статичка

ОДГОВОР: 5.

По дефиницији статички (static) атрибут – он је заједнички за све објекте класе.

163. У програмском језику C#, по дефиницији метод може да врати вредност, али и не мора. Изабрати службену реч која се користи за означавање типа резултата методе када дефинишемо метод који не враћа вредност:

1. void
2. return
3. public
4. static

ОДГОВОР: 1.

Службена реч која се користи за означавање типа резултата методе је **void** (у нашем случају). Пример методе:

public **void** Set(double xx, double yy) { x=xx; y=yy; }

public и **static** (одговор под 3. и 4.) су модификатори. **static** служе за дефинисање поља да ли ће бити статичко, ако се не наведе подразумева се да је поље објектно. Модификатор приступа **public** или јавно поље одређује где се поље може користити, конкретно се може користити у било којој класи.

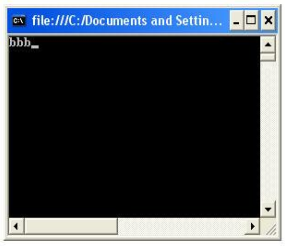
164. У програмском језику C# дат је метод *PrintPoruka* који је дефинисан на следећи начин:

```
void PrintPoruka(string poruka, int n) {
        while (n > 0) {
                Console.Write(poruka);
```

```
                n--;
            }
        }
```

Шта ће бити исписано на екрану након позива метода **PrintPoruka("b",3):**

1. На екрану се приказује:bbbbb
2. На екрану се приказује:bbbb
3. На екрану се приказује:bbb
4. На екрану се приказује:bb
5. На екрану се приказује:b



ОДГОВОР: 3.
While петља извршава се 3 пута, за n=3,2,1. После тестирања, резултат је приказан на слици.
Дакле, на екрану се приказује **bbb**

165. У програмском језику C# дат је метод PrintPoruka који је дефинисан на следећи начин:

```
    void PrintPoruka(string poruka, int n) {
            while (n > 0) {
                n--;
            Console.Write(poruka);
            }
    }
```

Израчунати вредност променљиве **m** након извршења следећег програмског фрагмента:

```
            int m = 2;
            PrintPoruka("C# !!!", m);
```

Заокружити број испред траженог одговора:

1. m =0
2. m =1
3. m =2
4. m =3

ОДГОВОР: 3.
Основни типови се преносе по вредности, не по референци, промена **m** у методи нема никаквог утицаја на "право" **m**.
Дакле, **m = 2** и после извршавања методе остаје **m = 2**.

166. Дата је наредба кода у програмском језику C# која дефинише константу MAX_CENA као чланицу неке класе. Који је од понуђених начина дефинисања исправан:

1. public static const MAX_CENA = 100.00;
2. public const float MAX_CENA = 99.50;
3. public static double MAX_CENA = 100.50;
4. public static const double MAX_CENA = 100.99;
5. public const double MAX_CENA = 100.99;

ОДГОВОР: 5.

1. Одговор није исправан: Недостаје тип константе нпр. **double** и не сме да се користи кључна реч **static.**
2. Одговор није исправан: Недостаје **f** тј. треба да пише **99.50f**
3. Одговор није исправан: Недостаје кључна реч **const** и не сме да се користи кључна реч **static**.
4. Одговор није исправан: Не сме да се користи кључна реч **static.**

167. Дата је дефиниција класе у програмском језику C# и састоји се од два конструктора, методе и пољах и y. У шестом реду написати конструктор копије објекта класе Point

```
1.      public class Point {
2.          private double x, y;
3.          public Point() { x = 0; y = 0; }
4.          public void Set(double xx, double yy){ x=xx; y=yy; }
5.          public Point(Point p) {
6.              _____//Odgovor
```

```
7.              }
8.      }
```

Заокружити број испред траженог одговора:

1. this(p.x, p.y);
2. this(p);
3. Set(p);
4. Set(p.x, p.y);

ОДГОВОР: 4.

Конструктор дефинисан одговором под 1 не постоји.
Конструктор дефинисан одговором под 2 би довео до бесконачне петље (јер смо већ у том конструктору).
Одговор под 3 покушава да позове методу која не постоји.
Set(p.x, p.y); је тачно, могли смо да напишемо и **x = p.x; y = p.y;**

```
public class Point {

        private double x, y;
        public Point() { x = 0; y = 0; }
        public void Set(double xx, double yy){ x=xx; y=yy; }
        public Point(Point p) {
            this(p.x, p.y);
            this(p);
            Set(p);
            Set(p.x, p.y);
        }
}
```

168. Дат је код програма у програмском језику C#. Анализирати дати код и проценити његову тачност:

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args) {
                        int n = 2;
                        fun(n);
                        Console.WriteLine("n je " + n);
                }
                void fun(int n) { n++; }
        }
}
```

1. Програм има грешку, јер метод fun() не враћа ниједну вредност.
2. Програм има грешку, јер метод fun() није дефинисан да буде статички.
3. Програм приказује 1 на екрану.
4. Програм приказује 2 на екрану.
5. Програм приказује 3 на екрану.

ОДГОВОР: 2.

fun() није статичка метода. Метода "main" је статичка. По C# стандарду, статичке методе могу да зову само друге статичке методе унутар своје класе, што "fun" није.

```
static void Main(string[] args)
{
        int n = 2;
        fun(n);
        Console.WriteLine("n je " + n);
        Console.Read();
}
void fun(int n) { n++; }
```

169. Дат је код програма у програмском језику C# који дефинише интерфејс Poredjenje. Интерфејс Poredjenje садржи декларацију методе porediPovrsine(), која пореди објекте типа Figura. У датом коду дописати у 3.линији кода наредбу која недостаје да би метода била исправно декларисана.

```
1.      namespace figure
2.
3.              public_____________________{
4.                      int porediPovrsine(Figura fig);
5.      }
```

Заокружити број испред одговора који даје исправно решење:

1. extends Poredjenje
2. interface Poredjenje
3. implements Poredjenje
4. abstract Poredjenje

ОДГОВОР: 2.

По C# стандарду, интерфејси се дефинишу са "**interface** ", па жељено име, дакле одговор је под 2 тј. **interface Poredjenje**

170. Дат је код програма у програмском језику C# којим су дефинисане две класе: class Program која садржи Main(string[] args) методу и classA. Анализирати дати код и одредити да ли је код исправно написан. Понуђени одговори дају опис последица извршавања овог кода:

```
class Program{
        public static void Main(string[] args){
                Object a1 = new A();
                Object a2 = new A();
                Console.WriteLine(a1.Equals(a2));
        }
}
class A{
        int x;
        public bool Equals(A a) {
                return this.x == a.x;
        }
}
```

1. Програм има грешку, јер се изразом a1.Equals(a2) проверава једнакост објеката a1 и a2 различитог типа од Object.
2. Програм има грешку, јер се једнакост објеката a1 и a2 типа A проверава изразом a1 == a2.
3. Програм се извршава без грешке и приказује се true на екрану.
4. Програм се извршава без грешке и приказује се false на екрану.

ОДГОВОР 4:
Класа **Object** дефинише нову методу у класи **A:** **public virtual bool Equals(Object a)**
Класа **A** има дефинисану методу: public bool Equals(A a)
Дакле, класа **A** дефинише методу "Equals", са погрешном листом параметара тј. дефинише некомпатибилну методу, која не надјачава оргиналну, самим тим **Object** не зна да она постоји, па онда користи своју уграђену тј. public bool Equals(A a).

У Main – у креирамо два објекта класе **A**, које су типа **Object** (ова операција сакрива све методе и поља која су везана за конкретну класу A). Сада када позовемо "Equals" (али који смо добили од Object), једино што компајлер види, онда проверава да ли су a1 и a2 иста ствар. Нема везе што можда имају исте вредности "int x", они су две различите ствари у меморији. Па ће резултат тога бити **false** испис на екрану.

171. Дат је код програма у програмском језику C# којим су дефинисане две класе: class Program која садржи Main(string[] args) методу и classA. Анализирати дати код и одредити да ли је код исправно написан. Понуђени одговори дају опис последица извршавања овог кода:

```
class Program{
        public static void Main(string[] args){
                A a1 = new A();
                A a2 = new A();
                Console.WriteLine(a1.Equals(a2));
        }
}
class A{
        int x;
        public bool Equals(A a){
                return this.x == a.x;
        }
}
```

1. Програм има грешку, јер се изразом a1.Equals(a2) проверава једнакост објеката a1 и a2 различитог типа од Object.
2. Програм има грешку, јер се једнакост објеката a1 и a2 типа A проверава изразом a1 == a2.
3. Програм се извршава без грешке и приказује се true на екрану.
4. Програм се извршава без грешке и приказује се false на екрану.

ОДГОВОР 3:
У овом случају, оргинална метода у класи A (а то је **public virtual bool Equals(A a)**) није надјачана, али то не прави проблем јер класа A зна све своје методе (за разлику од Object), па ће новодефинисана метода "Equals" бити позвана (а то је public bool Equals(A a)).

172. Једна од основних особина објектно оријентисаног језика је наслеђивање. Дате су насловне линије дефиниције класе. Заокружити број испред дефиниције класе која се не може наследити:

1. class A { }
2. class A { private A(){ } }
3. sealed class A { }
4. class A { protected A(){ } }

ОДГОВОР 3:
sealed class A { } - "sealed " значи "класа се не може наследити".

173. Дате су наредбе које су написане на C# програмском језику. Која наредба је исправно написана:

1. int a = Math.Abs(-5);
2. int b = Math.Abs(5.0);
3. int c = Math.Abs(5.5f);
4. int d = Math.Abs(5L);

ОДГОВОР 2:
Сви остали одговори враћају погрешне типове (double, float i long).

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.

ОДГОВОР 2:
По C# стандарду, исказ у одговиру под 2, није тачан (тражи се нетачна тврдња). Стандард такође каже да се приватним методама не може никако приступити ван класе, тако да "трикови" попут коришћења "base" неће упалити.

175. Дати су позиви методе Math.Max (метода враћа вредност већег броја) дефинисане у C# програмском језику. Анализирати дате позиве и одредити који од њих су исправно написани.
Заокружити број испред <u>неисправно</u> написаног позива методе:

1. Math.Max(1, 4);
2. Math.Max(2.3, 5);
3. Math.Max(1, 3, 5, 7);
4. Math.Max(-1.5, -2.8f);

ОДГОВОР 3:
Метода прима тачно два броја као аргументе, одговор под 3 је нетачан јер има 4 аргумента а не 2.

176. При креирању објеката изведене класе:

1. извршава се само конструктор изведене класе
2. прво се извршава конструктор родитељске класе, али само ако је позван кључном речју ***base***
3. обавезно се прво извршава конструктор изведене, а потом конструктор родитељске класе
4. обавезно се прво извршава конструктор родитељске, а потом конструктор изведене класе

ОДГОВОР 4:
Прво родитељске, па изведене - ово је увек тачно, с тим што ако родитељска класа има подразумевани конструктор, он се аутоматски позива. Ако нема, мора се ручно позвати жељени конструктор са "*base*" као прва наредба у изведеном конструктору.

177. Дат је део кода који је написан у C# програмском језику. Одредити шта ће се приказати на излазу:

```
try
{
        int x = 0;
        int y = 5 / x;
}
catch (Exception e)
{
```

```
        Console.WriteLine("Exception");
}
catch (ArithmeticException ae)
{
        Console.WriteLine(" Arithmetic Exception");
}
Console.WriteLine("finished");
```

1. Приказује се текст: finished
2. Приказује се текст: Exception
3. Ништа. Дешава се грешка приликом компајлирања
4. Приказује се текст: Arithmetic Exception

ОДГОВОР 3:
"Exception" већ покрива све могуће изузетке, па C# не дозвољава да се додају додатни (бескорисни) "catch" блокови.

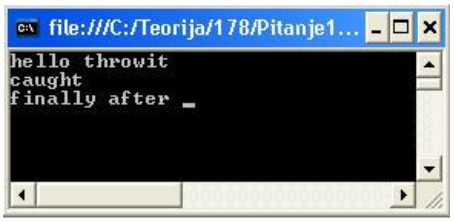
178. Дат је део кода који је написан у C# програмском језику. Одредити шта ће бити на излазу:

```
public class RTExcept
{
        public static void throwit() {
                Console.WriteLine("throwit ");
                throw new ApplicationException();
        }
        public static void Main(String[] args) {
                try {
                        Console.Write("hello ");
                        throwit();
                }
                catch (Exception re ) { Console.WriteLine("caught "); }
                finally { Console.Write("finally "); }
                Console.Write("after ");
        }
}
```

1. hello throwit caught
2. Грешка приликом компајлирања
3. hello throwit *RuntimeException* caught after
4. hello throwit caught finally after

ОДГОВОР 4:
По C# стандарду, редослед извршавања наредби је:

- наредбе пре try - catch - finally блока
- наредбе у try (док се не деси Exception)
- наредбе у catch (ако се деси Exception)
- наредбе у finally (у сваком случају)
- наредбе после try - catch - finally блока



179. Дат је код програма у програмском језику C#. Анализирати дати код и проценити његову тачност:

```
namespace TestPrimer{
        class Test {
                int x;
                public Test(string s){
                        Console.WriteLine("Klasa Test");
                }
                static void Main(string[] args){
                        Test t = null;
                        Console.WriteLine(t.x);
                }
        }
}
```

1. Програм има грешку, јер променљива x није иницијализована.
2. Програм има грешку, јер класа Test нема подразумевани конструктор.
3. Програм има грешку, јер се у некој класи не може декларисати променљива типа те исте класе, као што је то овде случај са променљивом t.
4. Програм има грешку, јер променљива t није иницијализована и има вредност null у моменту када се приказује поље t.x
5. Програм нема грешака и нормално се извршава, не приказујући ништа на екрану.

ОДГОВОР 4:
Први одговор није тачан – нема везе ако није x инцилизована, по default има вредност 0, и не би пријављивао грешку.
Други одговор није тачан – нема подразумевани конструктор али то не утиче на, пријављивање грешке.
Трећи одговор није тачан – у некој класи се може декларисати променљива те исте класе
Четврти одговор је тачан – програм има грешку , јер променљива t није иницијализована и има вредност null у моменту када се приказује поље t.x.

180. Дат је код програма у програмском језику C# који формира и штампа елементе низа a. Анализирати дати код и проценити шта ће се догодити након његовог извршавања.

```
namespace TestPrimer {
        class Program{
                static void Main(string[] args){
                        int[] a = new int[5];
                        for (int i = 0; i < a.Length; i++) a[i] = i;
                        Console.Write(a[i] + " ");
                }
        }
}
```

Заокружити број испред тачног одговора:

1. Програм приказује бројеве 0 1 2 3 4 на екрану.
2. Програм има грешку, јер ће у последњој наредби Console.**Write** метода **Main** покушати приступ непостојећем елементу a[5].
3. Програм приказује број 5 на екрану.
4. Програм има грешку, јер променљива ***i*** у последњој наредби Console.Write у методу Main неће имати дефинисану вредност.

ОДГОВОР 4:
Променљива i дефинисана у **for** наредби, постоји само унутар ове наредбе, тако да је програм неће видети ван ње и пријављиваће грешку.

181. Дат је код програма у програмском језику C#:

```
namespace TestPrimer{
        class Program
        {
                static void Main(string[] args)
                {
                        Console.WriteLine(fun(17));
                }
                public int fun(int n) { return n; }
                public void fun(int n){ Console.WriteLine(n); }
        }
}
```

Анализирати код и заокружити број испред очекиваног одговора:

1. Програм има грешку, јер се не може одредити коју верзију преоптерећеног метода **fun(…)** треба позвати.
2. Програм има грешку, јер је друга верзија преоптерећеног метода **fun(…)** дефинисана, али се нигде не позива.
3. Програм се нормално извршава и приказује 17 једанпут.
4. Програм се нормално извршава и приказује 17 двапут.

ОДГОВОР 1:
Не може се одредити шта се зове.

182. Дат је код програма у програмском језику C#. Анализирати програм и одредити шта се приказује на екрану као резултат његовог извршавања.

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args){

                        int[] x = { 0, 1, 2, 3, 4 };
                        Inc(x);
                        int[] y = { 0, 1, 2, 3, 4 };
                        Inc(y[0]);
                        Console.WriteLine(x[0] + " " + y[0]);
                }
                public static void Inc(int[] a){
                        for (int i = 0; i < a.Length; i++) a[i]++;
                }
                public static void Inc(int n) { n++; }
        }
}
```

1. Порука о грешци
2. 1 0
3. 2 2
4. 2 1
5. 1 1

ОДГОВОР 2:
Низови су референтни типови, па прва функција "inc" заправо мења низ. За све елементе низа повећава се вредност за један. Променљиве **int** су вредносног типа, па друга функција "inc" заправо не ради ништа.

183. Дат је код програма у програмском језику C# и састоји се од две класе у једној датотеци. Анализирати дати код и проценити која се вредност поља b.n приказује првом наредбом Console.WriteLine (ред седам) приликом извршавања овог програма.

```
1.      namespace TestPrimer{
2.              class Program{
3.                      static void Main(string[] args){
4.                              int k = 0;
5.                              Brojac b = new Brojac();
6.                              for (int i = 0; i < 100; i++) Inc(b, k);
7.                              Console.WriteLine("b.n = " + b.n);
8.                               Console.WriteLine("k = " + k);
9.                      }
10.                     public static void Inc(Brojac b, int k){
11.                             b.n++;
12.                             k++;
13.                     }
14.             }
15.             class Brojac{
16.                     public int n;
17.                     public Brojac(int n) { this.n = n; }
18.                     public Brojac() { this.n = 1; }
19.             }
20.     }
```

1. b.n = 101
2. b.n = 100
3. b.n = 99
4. b.n = 0
5. b.n = 1

ОДГОВОР 1:

Прво се , иницијализује променљива k (**k = 0**). Извршава се конструктор класе Brojac, где се иницијализује n тј. **b.n = 1.** Извршава се for петља, која 100 пута позива методу **Inc**. Метода повећава b.n за један сваки пут када се изврши (b.n++) али не и **k** (k = 0). Решење је под 1, исписује се **b.n = 101** и **k = 0**.

184. Дат је програм у програмском језику C# у ком је дефинисан низ и метода *Prosek* која израчунава средњу вредност низа. Анализирати дати код и проценити шта ће се десити након његовог извршавања.

```
namespace TestPrimer{
        class Program{
                static void Main(string[] args){

                        double[] niz = { 1.0, 2.0, 3.0 };
                        Console.WriteLine(prosek(niz));
                        Console.WriteLine(prosek(1, 2, 2, 1, 4));
                        Console.WriteLine(prosek(new double[] { 1, 2, 3 }));
                        Console.WriteLine(prosek(1.0, 2.0, 2.0, 1.0));
                }
                public static double prosek(params double[] brojevi){
                        double suma = 0;
                        foreach (double elem in brojevi) suma += elem;
                        return suma / brojevi.Length;
                }
        }
}
```

1. Програм има грешку у првој наредби WriteLine, јер је неправилан позив функције ***prosek (niz).***
2. Програм има грешку у другој наредби WriteLine, јер је неправилан позив функције ***prosek (1, 2, 2, 1, 4).***
3. Програм има грешку у трећој наредби WriteLine, јер је неправилан позив функције ***prosek(new double[] {1, 2, 3}).***
4. Програм има грешку у четвртој наредби WriteLine, јер је неправилан позив функције ***prosek(1.0, 2.0, 2.0, 1.0).***
5. Програм се извршава без грешке и просек датих бројева се тачно израчунава.
6. Програм се извршава без грешке, али се просек датих бројева не израчунава тачно.

ОДГОВОР 5:
Програм се изврашава, дакле нема грешку – пошто се добро позива функција **prosek** у сва три случаја. Можда делује да су други и трећи позив лоши, али стандард каже да се int може аутоматски претворити у double у скоро свим ситуацијама (једна од ретких где не може је приликом дефинисања статичких константи).
Ако се погледа функција prosek, може се видети да је исправно – правилно написана па се просек тачно извршава.

185. У програмском језику C# дата је декларација променљиве једне стринг и једне целебројне променљиве, као и део кода:

```
string str = "Primer";
int broj = 66;
Console.WriteLine(str + broj + 65);
Console.WriteLine(broj + 65 + str);
```

Анализирати код и проценити шта ће се приказати на екрану након његовог извршења:

1. **Primer6665**
   131Primer
2. Primer6665
   6665Primer
3. Primer131
   131 Primer
4. PrimerBA
   BAPrimer

ОДГОВОР 1:

Console.WriteLine(str + broj + 65); — **str** је типа стринг, па се зато broj и 65 третирају као два стринга (**Primer6665**) дакле, не сабирају се 66 и 65.

Console.WriteLine(broj + 65 + str); — пошто променљива broj није стринг, врши се операција сабирања 66 + 65= 131 и приказује стринг str. (**131Primer**)

Пример : Ако напишемо, Console.WriteLine(1 + 2 + str + 1 + 2); // програм ће исписивати **3Primer12**

186. На програмском језику C# дефинисане су класе:

```
public class A {
        public virtual int Metod() { return 10; }
}
public class B : A {
```

```
        public new int Metod() { return base.Metod() +20; }
}
public classC : B {
        public override int Metod() { return base.Metod() +30; }
}
```

Унутар функције Main, креирана су три објекта ових класа и из сваког од њих извршен позив методе ***Metod()***
Заокружити број испред понуђеног одговора који представља сценарио који ће се десити при покретању програма:

1. Метод позван из класе A враћа вредност 10, из класе B 10, а из класе C 30
2. Метод позван из класе A враћа вредност 10, из класе B 30, а из класе C 60
3. Програм се покреће, али баца изузетак при позиву методе из класе B jer new не иде у комбинацији са virtual
4. Компилатор јавља грешку јер у класи C метод не може бити редефинисан

ОДГОВОР 4:
Метода може бити редефинисана у наслеђеној класи са **override** само ако је у основној класи та метода virtual, abstract или већ редефинисана са override у односу на њену основну класу.
Овде имамо да је у класи B која наслеђује класу A дефинисана скоз нова метода:

```
public class B : A {
        public new int Metod() { return base.Metod() +20; }
```

А ова метода која није ни virtual ни abstract а ни override у односу на основну класу A, не може да се редефинише (override - је) у изведеној класи C. Дакле, **одговор је под 4**.
Другачије речено, у класи C компајер покушава да редефинише метод али не може зато што у класи B је метод означен са new уместо са virtual, abstract а или override и зато јавља грешку.

187. Дат је код у програмском језику C#, који дефинише рекурзивни метод. Анализирати код и одредити резултат извршавања задатог метода:

```
public long fun(int n){
        return n * fun(n - 1);
}
```

1. Резултат позива fun(3) је 1 .
2. Резултат позива fun(3) је 2.
3. Резултат позива fun(3) је 6.
4. Позив fun(3) изазива грешку јер производи бесконачан ланац позива истог метода fun(…).

ОДГОВОР 4:
Нигде не постоји услов прекида тако да се производи бесконачан ланац позива истог метода.

188. Дат је код у програмском језику C#, који дефинише рекурзивни метод. Анализирати код и одредити резултат који ће се приказати на екрану:

```
namespace TestPrimer {
        class Program{
                static void Main(string[] args){
                        fun(2);
                }
                public static void fun(int n) {
                        while (n > 1){
                                Console.Write((n - 1) + " ");
                                fun(n - 1);
                        }
                }
        }
}
```

1. Програм на екрану не приказује ништа
2. Програм на екрану приказује 1 2 3
3. Програм на екрану приказује 3 2 1.
4. Програм на екрану бесконачно приказује 1 1 1 1 1 ….
5. Програм на екрану бесконачно приказује 2 2 2 2 2 ….

ОДГОВОР 4:
Унутар позива првог "**fun**" вредност **n** се никад не мења, па ни **while** никад неће бити нетачан, да би се прекинуо бесконачан низ.

189. Дат је код у програмском језику C#, који дефинише рекурзивни метод. Анализирати код и одредити резултат извршавања задатог метода:

```
public static int fun(int n) {
        if (n == 1) return 1;
        else return n + fun(n - 1);
}
```

1. Позивом fun (4) се исти метод fun () позива још 2 пута.
2. Позивом fun (5) се исти метод fun () позива још 4 пута.
3. Позивом fun (4) се исти метод fun () позива још 4 пута.
4. Позивом fun (5) се исти метод fun () позива још 6 пута.

ОДГОВОР 2:
fun(5) позива fun() још 4 пута (укупно 5). Сваки пут се прослеђује број мањи за један. Дакле, **fun(5)** – први позив, **fun(4)** - још 1, **fun(3)** - још 2, **fun(2)** - још 3, **fun(1)** – још четири пута.

190. Дат је код у програмском језику C#, рекурзивни метод, који проверава да ли је неки стринг палиндром. Да би код био комплетиран потребно је допунити трећи ред условом if наредбе.

```
1.      public static bool palindrom(String s)
2.      {
3.              if (s.Length <= 1) return true; //bazni slučaj
4.              else if (_____) return false;
5.              else return palindrom(s.Substring(1, s.Length - 2));
6.      }
```

Заокружити број испред траженог одговора:

1. s[0] != s[s.Length - 1]
2. s[0] != s[s.Length]
3. s[1] != s[s.Length - 1]
4. s[1] != s[s.Length]

ОДГОВОР 1:
Одговор је под 1 тј. **s[0] != s[s.Length - 1]**. Зашто? Пореде се прво и последње слово, са споља ка унутра.
Коришћење рекурзивног метода за проверу да ли је неки стринг палиндром, на исти начин као овај пример може се видети на сајту http://digis.edu.rs/mod/page/view.php?id=2462 (трећи слајд, тамо је овај део написан на следећи начин:
s.charAt(0) != s.charAt(s.length() - 1)).

191. Дат је код у програмском језику C#, рекурзивни метод, који проверава да ли је неки string палиндром. Да би код био комплетиран потребно је допунити седми ред.

```
1.      public static bool Palindrom(String s){
2.              return Palindrom(s, 0, s.Length - 1);
3.      }
4.      public static bool Palindrom(String s, int levi, int desni){
5.              if (desni <= levi) return true; // bazni slucaj
6.              else if (s[levi] != s[ desni]) return false;
7.              else return_____;
8.      }
```

Заокружити број испред траженог одговора:

1. Palindrom(s)
2. Palindrom(s, levi, desni)
3. Palindrom(s, levi + 1, desni - 1)
4. Palindrom(s, levi + 1, desni)
5. Palindrom(s, levi, desni - 1)

ОДГОВОР 3:
Одговор је под 3 тј. **palindrom(s, levi + 1, desni - 1).** Зашто? Пореде се спољашња слова и онда треба померити и с леве и с десне стране ка средини. Ово се најбоље види из претходног примера (190. питање), погледати овај део кода:

palindrom(s.Substring(1, s.Length - 2))

192. Дат је код у програмском језику C#, рекурзивни метод за сортирање низа рационалних бројева у опадајућем редоследу. Да би код био комплетиран потребно је допунити други ред помоћу једног од понуђених одговора.

```
1.      public static void Sortiranje(double[] niz){
2.              _____________________________________;
3.      }
4.      public static void Sortiranje(double[] niz, int kraj){
5.              if (kraj > 0){
6.                      int imin = 0;
7.                       double min = niz[0];
8.                      for (int i = 1; i <= kraj; i++)
9.                              if (niz[i] < min){
10.                                     min = niz[i];
11.                                     imin = i;
12.                             }
13.                     niz[imin] = niz[kraj];
14.                     niz[kraj] = min;
15.                     Sortiranje(niz, kraj - 1);
16.             }
17.     }
```

Заокружити број испред траженог одговора:

1. Sortiranje(niz)
2. Sortiranje(niz, niz.length)
3. Sortiranje(niz, niz.length+1)
4. Sortiranje(niz, niz.length-1)

ОДГОВОР 4:
У осмом реду : **for (int i = 1; i <= kraj; i++)** , постоји део кода "**i <= kraj**", што значи да се дужина мора смањити за 1 да не би било прекорачења. Другачије написано, ако низ (niz) има 10 елемената, они су индексирани од 0 до 9, дужина низа је 10 па тај број треба смањити за један да не би било прекорачења.

193. Дата је дефиниција класе у програмском језику C#. Проценити где у дефиницији класе (испред које методе) треба заменити знакове ??? службеном речју static.

```
1.      public classTest {
2.              private int broj;
3.
4.              public ???  int kvadrant(int n) { return n * n; }
5.              public ??? int getBroj() { return broj; }
6.      }
```

Заокружити број испред тачне изјаве:

1. Метода kvadrant МОРА да буде статичка, док метода getBroj може и не мора.
2. Обе методе морају бити статичке.
3. Ни једна од дефинисаних метода није статичка.
4. Метода getBroj НЕ СМЕ да буде статичка, док метода kvadrant може и не мора.

ОДГОВОР 4:
Метода "**kvadrant**" се не ослања ни на шта, тако да може и не мора да буде статичка.
Метода "**getBroj**" се ослања на не статичко поље "broj" (погледати **private int broj;**), па не може бити статичка.

194. У програмском језику C#, метод *Equals()* за проверу једнакости два објекта је дефинисан у класи *Object*. У датом програмском коду, у класи *Klasa* је предефинисан метод *Equals()*. Анализирати код и проценити тачност извршења.

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args) {
                        Object obj1 = new Klasa();
                        Object obj2 = new Klasa();
                        Console.WriteLine(obj1.Equals(obj2));
                }
        }
        class Klasa {
                int x;
                public new bool Equals(Klasa o) { return this.x == a.x; }
        }
}
```

Заокружити број испред тачног одговора:

1. Програм има грешку, јер се изразом obj1.equals(obj2) проверава једнакост објеката obj1 и obj2 различитог типа од Object.
2. Програм има грешку, јер се једнакост објеката obj1 и obj2 типа ***Klasa*** проверава изразом obj1 == obj2.
3. Програм се извршава без грешке и приказује се ***true*** на екрану.
4. Програм се извршава без грешке и приказује се ***false*** на екрану.

ОДГОВОР 4:
Погледати 170. питање и одговор. Разлика је само што се у 170. питању користи класа "**A**" а у овом примеру "**Klasa**".
Напомена: Овај део (**return this.x == a.x;**) је штампарска грешка тако да би исправно било (**return this.x == o.x;**)

195. У програмском језику C#, метод *Equals(...)* је метод инстанце класе *Object* којим се проверавада ли је објекат из кога се метод позива једнак неком задатом објекту. Овај метод се може надјачати (override-овати) у наслеђеним класама. Одредити заглавље овог метода у класи *string* у којој би метод био надјачан.
Заокружити број испред понуђеног тачног одговора:

1. public override boolEquals(string s)
2. public new boolEquals(string s)
3. public override bool Equals(Object obj)
4. public static bool Equals(Object obj)
5. public boolEquals(string s1, string s2)

ОДГОВОР 3:
По C# стандарду одговор је под 3. Како то практично изгледа – погледати 196. питање и одговор.

196. У програмском језикуC#, метод *Equals()* за проверу једнакости два објекта је дефинисан у класи *Object*. У датом програмском коду у класи *Klasa* је надјачан (override-ован) метод *Equals()*. Анализирати код и проценити тачност извршења.

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args) {
                        Object obj1 = new Klasa();
                        Object obj2 = new Klasa();
                        Console.WriteLine(obj1.Equals(obj2));
                }
        }
        class Klasa {
                int x;
                public override bool Equals(object o) {
                        Klasa a = (Klasa) o;
                        return this.x == a.x;
                }
        }
}
```

Заокружити број испред тачног одговора.

1. Програм има грешку, јер се изразом obj1.equals(obj2) проверава једнакост објеката obj1 и obj2 различитог типа од Object.
2. Програм има грешку, јер се једнакост објеката obj1 и obj2 типа ***Klasa*** проверава изразом obj1 == obj2.
3. Програм се извршава без грешке и приказује се ***true*** на екрану.
4. Програм се извршава без грешке и приказује се ***false*** на екрану.

ОДГОВОР 3:
Погледати питање број **170.** Питања су слична – одговори су различити па је добро да се обрати пажња због могуће грешке. Овај пут је главна метода **public virtual bool Equals(Object o)** правилно надјачана од стане **public override bool Equals(object o)** методе. Тако да сад нема везе што пише "Object" јер и Object може да види правилно надјачане методе. Дакле, програм се извршава без грешке и приказује **true**.

197. Дат је код програма у програмском језику C#. У Main() методи декларисане су променљиве *s, obj* и *t*. Анализирати декларацију и одредити на који објекат указују променљиве *s, obj* и t.

```
namespace TestPrimer {
        class Program {
                static void Main(string[] args)
                {
                        string s = "c#";
                        object obj = s;
                        string t = (string) obj;
                }
        }
}
```

Заокружити број испред тачног одговора:

1. Када се вредност променљиве ***s*** додељује променљивој ***obj*** у наредби ***Object obj = s***, конструише се нови објекат.
2. Када се конвертује тип променљиве ***obj*** и њена вредност додељује променљивој ***t*** у наредби ***String t = (String)obj***, конструише се нови објекат.
3. Када се конвертује тип променљиве ***obj*** и њена вредност додељује променљивој ***t*** у наредби ***String t = (String)obj***, садржај променљиве ***obj*** се мења.
4. Променљиве ***s, obj*** и ***t*** указују на исти објекат типа **string**.

ОДГОВОР 4:
Све променљиве показују на исти објекат типа **string** и његова вредност је **"c#"**, дакле одговор је под 4.
Типови string и Object су референтни типови, па се приликом доделе, заправо мења референца. Тачније, **obj** се додељује референца променљиве **s**. (object obj = s;).

198. Дат је код програма у програмском језику C# који дефинише класу Osoba са њеним методама и атрибутима. Анализирати дати код и одредити који од понуђених одговора је потребно дописати у 1. ред кода како би метода била тачно дефинисана.

```
1.      public class Osoba_____________________________ {
2.              private string ime;
3.              private string prezime;
4.              int godina;
5.              public Osoba(string ime, string prezime, int godina) {
6.                      this.ime = ime;
7.                      this.prezime = prezime;
8.                      this.godina = godina;
9.              }
10.             public int GetGodina() { returnthis.godina; }
11.             public int CompareTo(Osoba obj) {
12.                     if (this.godina > obj.GetGodina()) return 1;
13.                     else if (this.godina < obj.GetGodina()) return -1;
14.                     else return 0;
15.             }
16.             public override string ToString() {
17.                     return"Ime: " + this.ime + "\tPrezime: " + this.prezime + "\tGodina: " + this.godina;
18.             }
19.     }
```

Заокружити број испред тачног одговора:

1. : IComparable<Osoba>
2. : IComparable
3. : IEquatable<Osoba>
4. : Comparer

ОДГОВОР 1:
Метода "**CompareTo(Osoba obj)**" – погледати 11. линију програмског кода, долази из "**IComparable<Osoba>**", није лоше погледати одговор на 201. питање, питања су слична – иста логика.

199. Датим кодом у програмском језику C# креира се пет објеката класе Osoba која имплементира интерфејс IComparable. Допунити код програма наредбом која, помоћу колекције података, формира генеричку листу особа - променљива lista и наредбу која врши сортирање те листе. Анализирати дати код и одредити који од понуђених одговора је потребно дописати у 7. и 10. линију кода како би се правилно декларисала и сортирала променљива lista.

```
1.      static void Main(string[] args) {
2.              Osoba a = new Osoba("Marko Ilic", 34);
3.              Osoba b = new Osoba("Mirko Prljic", 30);
4.              Osoba c = new Osoba("Danilo Sekara", 24);
5.              Osoba d = new Osoba("Sara Males", 15);
6.              Osoba e = new Osoba("Borko Ilic", 34);
7.              _____________________________________________ ;
8.              lista.Add(a); lista.Add(b); lista.Add(c);
9.              lista.Add(d); lista.Add(e);
10.             _____________________________________________ ;
11.             Console.WriteLine("Prikaz osoba po godinama starosti: ");
12.             foreach (Osoba x in lista) { Console.WriteLine(x); }
13.     }
```

Заокружити број испред тачног одговора:

1. List lista = new List(); и lista.Sort(null);
2. List<Osoba> lista; и lista.Sort();
3. ArrayList<Osoba> lista = new ArrayList<Osoba>(); и lista.Sort();
4. List<Osoba> lista = new List<Osoba>(); и lista.Sort();

ОДГОВОР 4:
Методи су дефинисани да треда тако да се зову: **List<Osoba> lista = new List<Osoba>();** и **lista.Sort();**

200. Дат је код програма у програмском језику C# који дефинише класу Osoba са њеним методама и атрибутима. Анализирати дати код и на основу декларације метода CompareTo (...) и Clone() одредити кôд који недостаје у првој линији.

```
1.      class Osoba : _____________________________________{
2.              private string ime;
3.              private double dohodak;
4.              public Osoba(string ime, double dohodak) {
5.                      this.ime = ime;
6.                      this.dohodak = dohodak;
7.              }
8.              public int CompareTo(Osoba osb) {
9.                      if (this.dohodak < osb.dohodak) return -1;
10.                     else if (this.dohodak > osb.dohodak) return 1;
11.                     else return 0;
12.             }
13.             public Object Clone() {
14.                     return this.MemberwiseClone();
15.             }
16.             public override string ToString(){
17.                     return"Ime: "+this.ime+"\nDohodak: "+this.dohodak;
```

```
18.             }
19.     }
```

Заокружити број испред тачног одговора:

1. IComparable<Osoba>, ICloneable
2. Comparable<Osoba>, Cloneable
3. IComparable<Osoba>, ICloneable<Osoba>
4. IComparable, ICloneable

ОДГОВОР 1:
Одговор је под 1 тј. прва линија програмског кода треба да изгледа: **class Osoba :IComparable<Osoba>, ICloneable {**
IComparable интерфејс, каже да класа која га имплементира мода да има методу:
public int CompareTo(**Object** osb)
С' обзиром да ова класа има методу: public int CompareTo**(Osoba** osb**)** то значи да класа није имплементирала генерални интерфејс **IComparable** неги конкретан **IComparable<Osoba>** .
ICloneable каже да класа имплементиира public Object Clone().
Другачије написано, метода "**CompareTo(Osoba osb)**" долази из "**IComparable<Osoba>**", а "**Clone**" из "**ICloneable**".

201. Дат је део кода који је написан на C# програмском језику. Анализирати и одредити шта ће се приказати на излазу извршавањем овог кода:

```
class PassA{
        public static void Main(string[] args){

                PassA p = new PassA();
                p.start();
        }
        public void start() {
                long[] a1 = {3,4,5};
                long[] a2 = fix(a1);
                Console.Write(a1[0] + a1[1] + a1[2] + " ");
                Console.WriteLine(a2[0] + a2[1] + a2[2]);
        }
        long[] fix(long[] a3){
                a3[1] = 7;
                return a3;
        }
}
```

1. 12 15
2. 15 15
3. 345 375
4. 375 375

ОДГОВОР 2:
Низови се прослеђују по референци, што значи да се метода **long[] fix(long[] a3),** променити елемент a3[1], тј. елементи низа биће **long[] a1 = {3,7,5};** и **long[] a2 = {3,7,5}**; . На екрану ће се прикати 15 15 (као сума елемената 3 + 7 + 5 = **15**).

202. Дат је код програма у програмском језику C# којим су дефинисане три класе: classProgram која садржи Main(string[] args) методу, classVozilo и classAuto. Компајлер приликом превођења овог кода јавља грешку. Заокружити број испред одговора у ком је написано који редови су изазвали грешку приликом компајлирања.

```
1.      class Program {
2.
3.              public static void Main(string[] args) {
4.                      Vozilo v = new Vozilo(8);
5.                      v.vozi();
6.                      v.brojVrata = 2;
7.                      Vozilo bmw = new Auto(2, 4);
8.                      bmw.vozi();
9.                      Auto audi = new Auto(4);
10.                     Auto.vozi();
```

```
11.                 }
12.         }
13.         class Vozilo {
14.                 public int brojTockova;
15.                 public Vozilo(int t) {
16.                         brojTockova = t;
17.                 }
18.                 public void vozi() {
19.                         Console.WriteLine("Vožnja vozila");
20.                 }
21.         }
22.         class Auto : Vozilo {
23.                 public int brojVrata;
24.                 public Auto(int v, int t) : base(t)
25.                 {
26.                         brojVrata = v;
27.                 }
28.                 public void vozi() {
29.                         Console.WriteLine("Vožnja auta");
30.                 }
31.         }
```

Заокружити број испред тачног одговора:

1. Грешке су у редовима 7, 10 i 1 6.
2. Грешке су у редовима 7, 9 i 22.
3. Грешке су у редовима 6, 7 i 10.
4. Грешке су у редовима 6, 9 i 10.
5. Грешке су у редовима 6, 9 i 24.
6. Грешке су у редовима 9, 10 i 24.

ОДГОВОР 4: 6, 9 и 10

6. линија програмско кода није тачна јер објекат **v** које је типа **Vozilo** нема поље **brojVrata**, има само **brojTockova.**
9. линија програмско кода није тачна јер конструктор **Auto** има два параметра а не један **тј.** **int v** и **int t**
10. линија програмско кода није тачна јер **vozi** није статичка метода**,** што значи да се мора позвати на нивоу објекта **auto.** Другачије речено**,** да смо написали **auto.vozi();** уместо **Auto.vozi();** компајлер не би приказивао грешку.

203. Дат је део кода који је написан на C# програмском језику. Анализирати код и одредити шта ће бити на излазу:

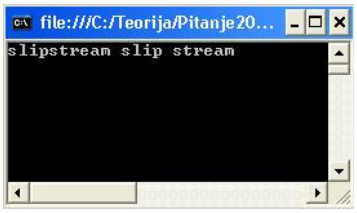
```
class PassS {
        public static void Main(String[] args){
                PassS p = new PassS();
                p.start();
        }
        void start(){

                String s1 = "slip";
                String s2 = fix(s1);
                Console.WriteLine(s1 + " " + s2);
        }
        String fix(String s1){

                s1 = s1 + "stream";
                Console.Write(s1 + " ");
                return"stream";
        }
}
```

1. slip stream
2. slipstream stream
3. stream slip stream
4. slipstream slip stream

ОДГОВОР 4: slipstream slip stream



- Позивањем методе **String fix(String s1),** на екрану се исписује **slipstream** јер је s1 = s1 + "stream" = "slip" + "stream".
- Кад се изврши **Console.WriteLine(s1 + " " + s2);** у оквиру методе **void start()** исписаће се **sleap stream** (s1 ="slip" и s2 = "stream")
- Закључак: Стрингови су референтни типови, али су такође **непроменљиви**, што значи да ће метода **fix** направити нови стринг у себи, али да неће променити прави s1.

204. Дат је код програма у програмском језику C# који дефинише апстрактну класу Figura. У класи Figura написати статички иницијални блок у ком се иницијализује вредност атрибута brojFigura. Заокружити број испред наредбе коју треба додати у 7.линију кода, којом се иницијализује вредност статичког атрибута brojFigura.

```
1.      public abstract class Figura {
2.              protected static int brojFigura;
3.              protected string naziv;
4.              public Figura(string naziv) {
5.                      this.naziv = naziv;
6.              }
7.              _____________________________________
8.              public abstract double povrsinaFigure();
9.              public abstract double obimFigure() ;
10.             public static void UvecajBrojFigura(){
11.                     brojFigura++;
12.             }
13.             public override string ToString(){
14.                     return"Figura ";
15.             }
16.     }
```

Заокружити број испред одговора који даје исправно решење:

1. static Figura() { brojFigura = 0; }
2. brojFigura = 0;
3. static { brojFigura = 0; }
4. public Figura() { brojFigura = 0; }

ОДГОВОР 1:
2. одговор није тачан јер иницијализија променљиве **brojFigura** се не може тако извршити.
3. одговор није тачан, има доста грешака, нпр. један од тих је, није наведен тип методе.
По C# стандарду, ако је класи потребан статички блок тј. **brojFigura,** а ово јесте статички блок (погледати 2. линију), потребно је да метода буде статичка, тј. **static Figura() { brojFigura = 0; }.** Дакле, решење је **одговор под 1**. Самим тим **није одговор под 4** јер је ово приватна метода а не статичка.

205. Дат је код програма у програмском језику C# у ком су дефинисане три класе: class Program која садржи Main(string[] args) методу, class A и class B. Анализирати дати код и одредити шта ће се приказати на екрану као резултат извршавања овог програма. Заокружити број испред одговора који садржи резултат исписа:

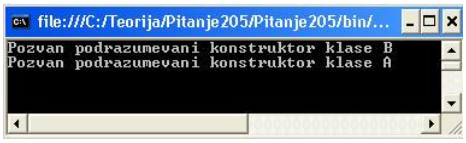
```
class Program : A {

        public static void Main(string[] args) {
                Program p = new Program();
        }
}
class A : B {
        public A() { Console.WriteLine("Pozvan podrazumevani konstruktor klase A"); }
}
class B {
        public B() { Console.WriteLine("Pozvan podrazumevani konstruktor klase B"); }
}
```

1. Ништа.
2. Позван подразумевани конструктор класе A
3. Позван подразумевани конструктор класе B
4. Позван подразумевани конструктор класе A и у другом реду: Позван подразумевани конструктор класе B
5. Позван подразумевани конструктор класе B и у другом реду: Позван подразумевани конструктор класе A

ОДГОВОР 5:

Ако погледамо програмски код видимо да је класа **Program** наслеђена од класе **A**, класа **A** је наслеђена од класе **B**, а класа **B** је најстарија класа. Конструктори се извршавају у супротном редоследу од наслеђивања, од "**најстаријег**" до "**најмлађег**". Па се прво извршава конструктор у класи **B**, па у класи **A** и на крају у класи **Program.**



206. Дат је код програма у програмском језику C# у ком су дефинисане три класе: class Program која садржи Main(string[] args) методу, class A и class B. Анализирати дати код и одредити да ли је код исправно написан. Заокружити број испред исказа који даје информацију о тачности кода:

```
class Program {
        public static void Main(string[] args) {
                B b = new B();
                b. Metod(5);
                Console.WriteLine("b.i je " + b.CitajI());
        }
}
class A {
        int i;
        publicint CitajI(){ return i;}
        public void Metod(int i) { this.i = i; }
}
class B : A {
        public voidMetod(string s){
                Console.WriteLine(s);
        }
}
```

1. Програм има грешку, јер је метод **Metod(inti)** надјачан (предефинисан) са различитим потписом у класи B.
2. Програм има грешку, јер се **b.Метод(5)** не може позвати пошто је метод **Metod(inti)** заклоњен у класи B.
3. Програм има грешку због **b.i**, јер је поље **i** неприступачно из класе B.
4. Програм нема грешке, јер наслеђени метод класе A,**Metod(inti)** није надјачан у класи B,већ је дефинисан преоптерећен метод **Metod(string s).**

ОДГОВОР 4:

Програм нема грешку, на екрану исписује **b.i je 5**, као што се види на слици.
Пошто метода у класи **A** прима тип **int,** а метода у класи **B** прима тип **string,** тј. типови су различити, ништа није заклоњено, па су оба метода доступна – **програм нема грешку**.

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

ОДГОВОР 4:

Према C# стандарду, каже се: **приватни чланови не могу се видети изван класе**.

208. Дата је наредба кода у програмском језику C# која представља декларацију низа. Које су декларација тачне:

1. int niz = new int(30);
2. double[ ] niz = new double[30];
3. int[ ] niz = { 3, 4, 3, 2 };
4. char[ ] niz = new char[ ];
5. char[ ] niz = new char { 'a', 'b', 'c', 'd' };
6. char[ ] niz = new char[ ] { 'a', 'b' };

ОДГОВОР 2,3 и 6:
Први одговор није тачан због малих заграда (тј. исправно би било **int niz = new int[30];**). Четврти одговор није тачан јер није одређено колико низ има елемената, правилно би и било нпр. **char[ ] niz = new char[10];.** Шести одговор није тачан јер је потребно ставити велике заграде, правилно би било, **char[ ] niz = new char []{ 'a', 'b', 'c', 'd' };**

209. Да би наслеђени метод могао да се редефинише и тиме измени његова функционалност у класама наследница ма, у родитељској класи испред ознаке повратног типа метода наводи се нека од понуђених кључних речи. Заокружити бројеве испред кључних речи које омогућавају редефинисање дефинисаног метода кроз ланац наслеђивања:

1. new
2. virtual
3. sealed
4. override
5. abstract
6. base
7. довољно је да буде public или protected

ОДГОВОР 2 , 4 и 5:
Одговор **virtual, override** и **apstract** – користе се за редефинисање наслеђеног метода. Погледати питање 228, у поставци овог питања (228) је одговор на питање (209).

210. У класи Figura дат је подразумевани (default) конструктор и конструктор са 4 параметра:

```
public Figura() { ... }
public Figura(string ime, string boja, int pozX, int pozY) { ... }
```

Заокружити бројеве испред исправно написаних наредби креирања објеката класе Figura:

1. `Figura f = Figura("lovac", "beli", 7, 3);`
2. `Figura f = new Figura("beli", "lovac", 7, 3);`
3. `Figura f = new Figura();`
4. `Figura f = new Figura("lovac", 3, 7, "beli");`
5. `Figura f = new Figura("lovac", "beli", 3, 7);`
6. `Figura f = new Figura("lovac", "beli", 3);`

ОДГОВОР 2, 3 и 5:
Figura има два конструктора. Један је празан, а други са четири параметра и то два стринга и два броја. Исправно су написане наредбе које могу да се напишу задавољавајући тај шаблон. Дакле **први одговор је нетачан** јер није добро креиран објекат (потребно написати **new**), четврти одговор нетачан због распореда аргумената ("lovac", 3, 7, "beli") и шести одговор нетачан јер су наведена 3 аргумента.

211. Дат је код на C# - у којим су креиране три класе у ланцу наслеђивања. Имајући у виду класификаторе приступа пољима класа, заокружити бројеве испред поља која ће бити видљива унутар класе Sin:

```
public class Deda {
        private double penzija;
        protected string adresa;
        public string ime;
}
public class Otac: Deda {
        private double plata;
        protected string struka;
}
public class Sin: Deda {
        public int razred;
}
```

1. penzija
2. **adresa**
3. **ime**
4. plata
5. **struka**
6. **razred**

ОДГОВОР 2, 3, 5 и 6:
Класа "Sin" види сва **public** и **protected** поља из класе које наслеђује, дакле:

- `protected string adresa;`
- `public string ime;`
- `protected string struka;`

Наравно, код себе види сва поља, дакле:

- public int razred;

НАПОМЕНА: Овде је учињена штампарска грешка, при формирању ланца наслеђивања, уместо **public class Sin: Deda** потребно је да пише **public class Sin: Otac**

212. Дат је код на C#- у којим су креиране три класе у ланцу наслеђивања. Унутар сваке класе декларисан је по један private, public и protected атрибут. У методи *Main()* класе *Program* креиран је објекат s класе *Sin* (Sin s = new Sin();) Заокружити бројеве испред поља која ће бити видљива у креираном објекту s класе Sin:

```
public class Deda {
        private double penzija;
        protected string adresa;
        public string ime;
}
publicclassOtac: Deda {
        private double plata;
        protected string firma;
        public string struka;
}
publicclassSin: Otac {
        private double prosek;
        protected int razred;
        public string skola;
}
```

1.penzija
2. adresa
3. ime
4. plata
5. struka
6. firma
7. prosek
8. razred
9. skola

ОДГОВОР 3, 5 и 9:
За некога ко посматра претходно и ово питање. Претходно питање, једноставније речено је гласило: "Које променљиве класа Sin види унутар себе?" и ту је одговор наведен. У овом задатку питање гласи "Шта неко види ко користи објекте класе Sin?" Објекти виде само **public** поља, дакле: **skola**, **struka** и **ime**.

213. Дати су искази који се односе на правила писања try-catch-finally блокова за руковање изузецима. Који искази су тачни:

1. Блок try мора имати бар један catch блок
2. Блок try може имати више catch блокова
3. Ако блок try има више catch блокова, изузетак основне Exception класе мора се хватати у првом catch блоку
4. Ако блок try има више catch блокова, битан је редослед њиховог писања
5. Блок try мора имати бар један finally блок
6. Блок try не сме да има више catch блокова

ОДГОВОР 1,2 и 4:
По C# стандарду, ако блок try има више catch блокова, основни Exception мора се хватати на крају (**3. Одговор нетачан**), try блок не мора да има finally блок (**5 одговор нетачан**) и try блок може да има више catch блокова (**6.одговор нетачан**).

214. Који искази у програмском језику C# дефинишу конструктор:

1. Подразумевани конструктор без параметара се увек аутоматски додаје класи.
2. Подразумевани конструктор без параметара се класи аутоматски додаје уколико у њој није експлицитно дефинисан ниједан конструктор.
3. У класи се мора експлицитно дефинисати бар један конструктор.
4. Конструктори немају тип резултата, чак ни **void.**

ОДГОВОР 2 и 4:
1. одговор није тачан – јер је половично тачан, прецизнији и тачан је одговор под 2.
3. одговор није тачан - јер немора експлицитно бити дефинисан бар један конструктор.

215. Дати су искази који дефинишу полиморфизам, једну од особина објектно-оријентисаног језика C#. Који искази су тачни:

1. Када је класа Б изведена из класе А, објекту класе А може се доделити показивач на објекат класе Б.
2. Када је класа Б изведена из класе А, у генеричку листу објеката класе Б, могу се убацити и објекти класе А и објекти класе Б.

3. Објекат типа Б се може пренети као аргумент методу на место параматера типа А уколико је Б класа наследница од А.
4. Када се у класи Б која је изведена из класе А изврши предефинисање (override) наслеђене методе, онда су у објекту класе Б доступне обе верзије ове методе (исто као код преоптерећивања метода – overload).

ОДГОВОР 1 и 3:
2. одговор није тачан, C# је одличио да избаци функционалност из овог одговора, зато што је често доводила до грешке.
4. одговор није тачан, у C# су само методе означене са "virtual" доступне и у наслеђеним класама.

216. Дате су наредбе у програмском језику C# које дефинишу заглавље методе Print() са променљивим бројем параметара. Одредити који од понуђених одговора су исправни:

1. public void Print(params string[] niska, params double[] broj)
2. public void Print(params double[] broj, string niska)
3. public void params Print(double d1, double d2)
4. public void Print(params double[] broj)
5. public void Print(int n, params double[] broj)

ОДГОВОР 4 и 5:
Према C# стандарду. Низовни параметар (double[], string[], …) мора бити последњи параметар у методи, па самим тим може постојати само један, такође не може бити повратна вредност, већ само улазни параметар. Дакле,
1. одговор није тачан јер има 2 низовна параметра – **params string[] niska, params double[] broj**,
2. одговор није тачан јер низовни параметар није последњи у методи – **params double[] broj, string niska**
3. одговор није тачан јер не може бити повратна вредност - public void **params** Print(double d1, double d2)

217. Дат је код програма у програмском језику C#. Код садржи објекте две класе у којима је дефинисан метод *toString()*. Анализирати код датог програма и одредити који од датих исказа су тачни.

```
namespace TestPrimer{

        class Program {
                static void Main(string[] args) {
                        Object a = new KlasaA();
                        Object obj = new Object();
                        Console.WriteLine(a);
                        Console.WriteLine(obj);
                }
        }
}
        class KlasaA{
                int x;
                public override string ToString() { return"x u A je " + x; }
        }
```

Заокружити бројеве испред очекиваних одговора:

1. Програм има грешку, јер наредбу ***Console.WriteLine(a)*** треба заменити наредбом ***Console.WriteLine(a.toString()).***
2. Приликом извршавања наредбе ***Console.WriteLine(a)***, програм позива се метод toString() наслеђен из класе Object.
3. Приликом извршавања наредбе ***Console.WriteLine(a)***, програм позива метод toString() из класе KlasaA.
4. Приликом извршавања наредбе ***Console.WriteLine(obj)***, програм позива метод toString() из класе Object.

ОДГОВОР 3 и 4:
Метода **"ToString"** је правилно надјачана, што значи да је и **Object** види.
Први одговор је нетачан, јер се **"ToString"** позива аутоматски у овом случају.
Други одговор је нетачан, јер је метода правилно надјачана.

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 {
        class Program {
                static void Main(string[] args) {
                        KlasaB b = new KlasaB();
                        b.Print();
```

```
        }
    }
    class KlasaA {
        string s;
        public Klasa A(string s) { this.s = s; }
        public void Print() { Console.WriteLine(this.s); }
    }
    class KlasaB : 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***

ОДГОВОР 2 и 3:
Пошто класа B нема ни један контруктор, аутоматски се генерише подразумевани. Па компајлер **класу B** види:

```
class KlasaB : KlasaA
{
    public KlasaB() : base()
    { }
}
```

Међутим "base()" покушава да позове конструктор "A()" који не постоји, што доводи до грешке.
Први одговор није тачан: јер као што смо написали класа B већ има подразумевани конструктор, само није ручно написан.
Четврти одговор није тачан, јер програм има грешку.

219. Дат је код програма у програмском језику C# који декларише два објекта типа Pravougaonik и један објекат типа Krug. Класе Pravougaonik и Krug наслеђују апстрактну класу Figura и обе класе имају методуToString. Код програма треба допунити са наредбом која ће исписивати низ Figura[] декларисан у коду програма.

```
1.  public abstract class Figura {
2.      protected static int brojFigura=0;
3.      public static int BrojFigura { get{ return brojFigura;}}
4.      public Figura() {brojFigura++; }
5.      public abstract double povrsinaFigure();
6.      public override string ToString(){ return"Figura ";}
7.  }
    …
1.  Pravougaonik pr1 = new Pravougaonik();
2.  Pravougaonik pr2 = new Pravougaonik(3, 2);
3.  Krug kr1 = new Krug(2);
4.
5.  Console.WriteLine("Broj figura="+Figura.BrojFigura+"\n");
6.  Figura[] niz = new Figura[ Figura.BrojFigura];
7.  niz[0] = pr1; niz[1] = pr2; niz[2] = kr1;
8.
9.  Console.WriteLine("Prikaz figura:");
10. _____
```

Заокружити бројеве испред одговора који представљају могуће начине приказа низа фигура:

1. foreach(Figura el in niz) Console.WriteLine(el);
2. for(int i=0; i<niz.Length; i++) Console.WriteLine(niz[i]);
3. foreach (el in niz) Console.WriteLine(el);
4. foreach (int i in niz) Console.WriteLine(niz[i]);

ОДГОВОР 1 и 2:

Овај задатак има гомилу кода, а цело питање се своди на "како се правилно пише for наредба?". Одговори под 1 и 2 су тачни. **Трећи одговор** је нетачан јер за променљива **el** нема тип променљиве – недостаје **Figura**. Четврти одговор је у потпуности нетачан, погледати како изгледа 1. одговор.

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{
        class Program{
                static void Main(string[] args){
                A a = new A();
                a. Print();
                }
        }
        class A{
                string s;
                public A(string s) { this.s = s; }
                public void Print(){
                        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");**.

ОДГОВОР 2 и 4:
1. Одговор је нетачан – класа A је јавна класа.
2. Одговор је тачан – класа A нема подразумевани конструктор, он се аутоматски додаје само ако не постоји ни један други. Пошто класа A већ има има конструктор он се неће додати.
3. Одговор је нетачан јер програм има грешку.
4. Одговор је тачан ако се наредба **A a = new A();** замени наредбом **A a = new A("poruka");**

221. Дата је дефиниција класе у програмском језику C# и састоји се од два конструктора, једне методе и поља x. У дефиницији се користи службена реч this. Анализирати дати код и проценити тачност следећих исказа:

```
class TestPrimer{
        public double x;
        public TestPrimer(double x){
                this.Fun();
                this.x = x;
        }
        public TestPrimer() {
                Console.WriteLine("Podrazumevani konstruktor");
                this(23) ;
        }
        public voidFun(){
                Console.WriteLine("Poziv metoda fun()");
        }
}
```

1. this.Fun() у конструктору TestPrimer(double x) може се поједноставити и заменити само са Fun().
2. this.x у конструктор у TestPrimer(double x) може се поједноставити и заменити само са x.
3. позив конструктора this(23) унутар другог конструктора TestPrimer() је прво шта се извршава и мора се писати одмах после декларације public TestPrimer(): this(23)
4. this(23) у конструктору Test() мора се заменити са прецизнијим изразом this(23.0).

ОДГОВОР 1 и 3:
1. одговор је тачан, C# стандард каже да се **"this."** Може уклонити ако ће наредба и даље остати једнозначна. У овом случају, постоји само једна метода **"fun"** (управо она из класе), па је једнозначна да се она позива.
3. одговор је тачан, C# стандард каже да се позив конструктора (родитељског или братског) мора да буде прва наредба.

222. Дата је дефиниција класе у програмском језику C# и састоји се од два конструктора, методе и поља x и y. У петом реду дефинисан је конструктор са параметрима који формира тачку са координатама x и y. Допунити дефиницију конструктора.

```
1.      public class Point {
2               private double x, y;
3.              public Point() { x = 0; y = 0; }
4.              public void set(double xx, double yy) { x = xx; y = yy; }
5.              public Point(double x, double y) { _____________; }
6.      }
```

Заокружити бројеве испред тачних одговора:

1. this.x=x; this.y=y;
2. x=x; y=y;
3. set(x,y);
4. set(this.x,this.y);
5. x=this.x; y=this.y;

ОДГОВОР 1 и 3:
2. одговор је нетачан јер је потребно **"this.__"**
4. одговор је нетачан јер у "this.x" и "this.y" треба да се упишу вредности, не да се читају из њих.
5. одговор је нетачан, исто као под 4.
Напомена: Погледати следеће питање и видети шта је написано, за овај део за који се тражи одговор.

223. Дата је дефиниција класе у програмском језику C# и састоји се од два конструктора и поља x и y. У четвртом реду допунити подразумевани конструктор без параметара класе Point, који формира тачку у координатном почетку, позивајући конструктор са параметрима.

```
1.      public class Point {
2.              private double x, y;
3.              public Point(double x, double y) {
4.                      this.x = x; this.y = y;
5.              }
6.      public Point() _____________
7.      }
```

Заокружити бројеве испред тражених одговора:

1. public Point() : base(0, 0) { }
2. public Point() : this(0, 0) { }
3. public Point() { Point(0,0); }
4. public Point() : this(0.0, 0.0) { }
5. public Point() { Point(0.0, 0.0); }

ОДГОВОР 2 и 4:
C# дозвољава да се конструктори ослањају једни на друге, тј. да један контрукор зове други. То се ради преко **": this()" синтаксе. У овом примеру** Point() се ослања на Point(double x, double y), па се пише **"Point() : this(0, 0)"** или **"Point() : this(0.0, 0.0)".**

224. Дати су делови кода у програмском језику C# који треба да рачунају збир елемената матрице a, декларисане на следећи начин: int[,] a = new int[10, 10] . Анализирати дате кодове и проценити који од предлога је тачан.

1.
```
int sum = 0;
for (int i = 0; i < b.Length; i++)
for (int j = 0; j < b[i].Length; j++)
        sum3 += b[i][j];
```
2.
```
int sum = 0;
foreach (int x in a) sum1 += x;
```

3.
```
int sum = 0;
```

```
for (int i = 0; i < a.GetLength(0); i++)
for(int j=0; j<a.GetLength(1); j++)
        sum2 += a[i,j];
```

4.
```
int sum = 0;
foreach (int[] vrsta in b)
foreach (int el in vrsta)
        sum4 += el;
```

ОДГОВОР 2 и 3:
Код одговора 2, користи се један for да се прође кроз све елементе матрице. C# је оргинално имао матрице, као јава int[][], али је после додата нова верзија int[,] која је унела нека побољшања, као што је нпр. Пролазак кроз све елементе одједном.
Одговор под 3 је класични for, где се пролази спољашњим for – ом, кроз редове, унутрашњим кроз колоне. GetLength метода враћа дужину матрице на n – тој позицији. Нпр. int[,] a = new int[5, 10] → a.GetLength(0) == 5, a.GetLength(1) == 19
Одговор под 1 није тачан, јер користи погрешну синтаксу [i][j] уместо [i, j].
Одговор под 4 није тачан, јер покушава у првом for – у да прође кроз све низове који су део матрице, а онда кроз све елементе који су део сваког низа.

225. На програмском језику C# дефинисане су класе:

```
1.      public class A {
2.              public virtual int Metod() { return 10; }
3.      }
4.      public class B : A {
5.              public new int Metod() { return 20; }
6.      }
7.      public class C : B {
8.              public override int Metod() { return 30; }
9.      }
```

Компилатор јавља грешку при превођењу овог кода коју је могуће решити на више начина у зависности од очекиваног ефекта.
Која ће решења отклонити грешку у коду:

1. У 5. линији кода метод у класи B прогласити за abstract уместо new
2. У 5. линији кода кључну реч new заменити са override
3. У 7. линији кода класу C наследити из класе A, уместо из класе B
4. У 8. линији кода, иза декларације методе у класи C, позвати основни метод :base()
5. У 8. линији кода обрисати кључну реч override и заменити је са sealed

ОДГОВОР 2 и 3:
1. одговор није тачан, јер метод у класи B, мора почети као апстрактна, не може се тек тако у некој "десетој" класи прогласити апстрактном.
4. одговор није тачан јер "**:base()**" уопште није валидан код.
5. одговор није тачан јер не уклања основни прлоблем: да класа B спречава даље редефинисање.

226. Дат је код програма у програмском језику C# којим су дефинисане четири класе: classProgram која садржи Main(String[] args) методу, classMasterStudent, classStudent и classOsoba. Приликом превођења овог кода компајлер јавља грешку.
Заокружити бројеве испред понуђених исказа који описују разлоге настајања грешке приликом компајлирања:

```
class Program{
        static void Main(string[] args){
                m(new MasterStudent());
                m(new Student());
                m(new Osoba());
                m(new Object());
        }
        public static void m(Student x){
                Console.WriteLine(x.ToString());
        }
```

```
}
class MasterStudent : Student{ }
class Student: Osoba{
        public string toString() { return"Student";}
}
class Osoba: Object{
public string toString(){ return"Osoba";}
}
```

1. Грешка је у позиву m(new MasterStudent()).
2. Грешка је у позиву m(new Student()).
3. Грешка је у позиву m(new Osoba()).
4. Грешка је у позиву m(new Object()).

ОДГОВОР 3 и 4:
Одговори под 1 и 2 немају грешку, јер метода **m** прима само објекте **"Student"** или које наслеђују од **"Strudent"**. То су класе **Student** и **MasterStudent.** Osoba и Object не испуњавају услов.

227. Преко јавних својстава (public property) може се остварити приступ приватним члановима класе.

Постављање вредности приватног поља врши се наредбом ___**set**_____, док се *читање* вредности приватног поља врши наредбом ____**get**____ у оквиру јавног својства.

228. Метод дефинисан у родитељској класи, у класама наследницама може бити редефинисан или сакривен.

Да би се омогућило **редефинисање** методе, при дефиницији у родитељској класи, испред ознаке повратног типа метода наводи се кључна реч **virtual**, **abstract** или **override**, док у изведеној класи испред ознаке повратног типа треба навести кључну реч ___**override**___.
Сакривање методе родитељске класе врши се тако што се у изведеној класи испред ознаке повратног типа наведе кључна реч ___**new**___.

229. Започете су изјаве које се односе на делове кода за обраду изузетака.

Довршити започете реченице:

Наредбе које се извршавају у случају настанка грешке, стављају се унутар блока	_____catch_____
Наредбе које се извршавају и ако се деси и ако се не деси грешка, стављају се унутар блока	_____finally_____
Наредбе које могу изазвати грешку стављају се унутар блока	_____try_____

230. На програмском језику C# дефинисана је структура *SVreme* и класа *CVreme*, а затим креиране по две промењиве типа *Svreme* и *CVreme*:

```
public struct SVreme {
        public int sat;
        public int min;
        public SVreme(int h, int m){
                sat = h; min = m;
        }
}
public class CVreme {

        public int sat;
        public int min;
        public CVreme(int h, int m) {
                sat = h; min = m;
        }
}
SVreme sv1 = new SVreme(12,15);
SVreme sv2 = sv1;
CVreme cv1 = new CVreme(12,15);
CVreme cv2 = cv1;
```

По извршењу следећих наредби доделе вредности пољима променљивих st1 и cv1, одредити и на одговарајућу линију уписати вредности које ће имати одговарајућа поља променљивих st2 и cv2.

sv1.sat = 24; sv2.sat = _____12_____;
cv1.sat = 24; cv2.sat = _____24_____;

Објекти класа су референтни типови, тако да су им поља иста. Док, то није случај са структуром.

231. На програмском језику C# дефинисане су две класе:

```
public class Roditelj {

        public virtual void Poruka1() { Console.WriteLine("R1"); }
        public void Poruka2() { Console.WriteLine("R2"); }
}
public class Dete: Roditelj {

        public override void Poruka1(){ Console.WriteLine("D1"); }
        public new void Poruka2(){ Console.WriteLine("D2"); }
}
```

Унутар функције Main, креирана су два објекта ових класа на следећи начин:

```
Dete x = new Dete();
Roditelj y = new Dete();
```

Проценити ефекат извршења наведених позива, и на предвиђене линије уписати шта ће се видети на стандардном излазу извршењем позваних метода:

x.Poruka1(); ___D1____
x.Poruka2(); ___D2____
y.Poruka1(); ___D1____
y.Poruka2(); ___R2____

ОДГОВОР:

У C# - у "укупни" тип неке променљиве се састоји од два дела:

- **декларисани тип** (декларисани тип је тип под којим променљива "гледа" на објекат, променљива **x** има декларисани тип **"Dete"** а променљива **y** има **"Roditelj"**).
- **стварни тип** (то је тип који објекат заправо има, променљиве **x** и **y** обе имају стваран тип "**Dete**").

Виртуелне методе ("virtual") се везују за стварни тип, а остале за декларисани.
Први и други одговор је једноставан, позивају се методе **Poruka1()** и **Poruka2()** у класи **Dete** па је одговор D1 и D2.
Објашњење за треће питање: Метода **Poruka1()** је виртуелна метода, па се везује за стварни тип променљиве y а то је **"Dete"** па се исписује порука **D1.**
Објашњење за четврто питање: Метода **Poruka2()** није виртуелна метода, па се везује за декларисани тип променљиве y а то је **"Roditelj"** па се исписује порука **R2**.

232. На програмском језику C# дефинисане су две класе:

```
public class KlasaA {
        public virtual int Metod() { return 10; } }
public class KlasaB : KlasaA {
        public override int Metod() { return 20; } }
public class KlasaC : KlasaB {
        public new int Metod() { return 30; } }
```

Креирани су објекти ових класа и из њих позвана метода ***Metod().*** На предвиђене линије уписати шта метод ***Metod()*** враћа при позиву из наведених објеката:

KlasaA a = new KlasaA(); a.Metod()	враћа вредност	___10___
KlasaB b = new KlasaB(); b.Metod()	враћа вредност	___20___
KlasaA bb = new KlasaB(); bb.Metod()	враћа вредност	___20___
KlasaC c = new KlasaC(); c.Metod()	враћа вредност	___30___
KlasaB cc = new KlasaC(); cc.Metod()	враћа вредност	___20___
KlasaA ccc = new KlasaC(); ccc.Metod()	враћа вредност	___20___

ОДГОВОР: 10, 20, 20, 30, 20, 20
За **први** и **други одговор** је јасно да ће се позвати метода public virtual int Metod() { return 10; } } и public override int Metod() { return 20; } }, и да је враћена вредност **10** и **20**. Исто важи и **четврти одговор**, дакле враћена вредност је **30**. Пошто се виртуелне методе се везују за стварни тип, онда је **трећи одговор**, **20**.
Пети и шести одговор је 20 и 20. Метод у **класи C** није виртуелан или override, што значи да више није видљив када гледамо у објекат кроз KlasaA и KlasaB. Видљив је само онај у **KlasaB.**

233. Са леве стране наведене су врсте грешака које се могу десити у коду, а са десне описи тих грешака. На линију испред описа уписати редни број под којим је наведен одговарајући тип грешке:

1. Синтаксне грешке (syntax error) — **1** — Откривају се у фази компилације и најлакше су за кориговање
2. Логичке грешке (logical error) — **3** — Не морају, али се могу десити током извршења и тада доводе до „пуцања“ програма
3. Грешке у време извршења (run-time error) — **2** — Програм функционише, нема „пуцања“, али се добијају погрешни резултати

234. Са леве стране дате су кључне речи које одређују типове класа, а са десне су описи класа. На линију испред описа уписати редни број под којим је наведен одговарајући тип класе:

1. abstract — **3** — Класа која се простире у више фајлова
2. sealed — **4** — Класа садржи само декларације метода, али не и дефиницију (тело) методе
3. partial — **1** — Класа која се не може инстанцирати
4. interface — **2** — Класа из које се не може наслеђивати

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

236. Са леве стране су наведени делови/елементи класе, а са десне стране улоге појединих класних елемената. На линују испред описа улоге унети редни број под којим је наведен одговарајући елеменат класе:

1. поље (атрибут) — **4** — Опис функционалности објеката класе
2. деструктор — **5** — Контрола приступа пољима класе
3. конструктор — **1** — Опис особина објеката класе
4. метод — **3** — Креирање објеката класе
5. својство / property — **2** — Уништавање објеката класе

237. Са леве стране дате су врсте променљивих у програмском језику C#, а са десне декларације променљивих. На линију испред декларације променљиве унети број којим је означена одговарајућа врста променљиве:

1. Променљива вредносног типа
2. Променљива референтног типа
3. Именована константа

3 — const double epso = 8.85E-12
1 — int x = 10;
2 — int[] a = new int[7];
2 — int[,] b=new int[10,10];
2 — int[] c = {1,2,3,4,5,6,7};

ОДГОВОР:
1. Променљивим вредносног типа припадају: прости типови података (нпр. Int, float, double) и структуре. **int x = 10;**
2. Променљивим референтног типа припадају стринг, објекат, класе, интерфејси, низови и делегати.
int[] a = new int[7]; int[,] b=new int[10,10]; int[] c = {1,2,3,4,5,6,7};
3. Очигледно је да **именованој константи** одговара **const double epso = 8.85E-12**, због кључне речи const.