Садржај

Предговор

Књига је намењена свима које интересује објектно оријентисано програмирање, развој класа и рад са објектима у креирању апликација.

Да би могао да се прати садржај изложен у овој књизи, неопходно је основно познавање програмирања. Подразумева се познавање неког језика који има C синтаксу. Иако ће у уводном поглављу бити објашњени и неки једноставни примери кроз обнављање, књига се не бави детаљно типовима података, декларацијама променљивих, операторима, изразима, наредбама за контролу тока апликације и осталим темама које су најчешће покривене неким основним курсом програмирања, као што су и алгоритми за претрагу, издвајање екстремних вредности, сортирање и сл. Читаоци треба да имају довољно знања да разумеју написани програмски код и ток апликације.

Књигу могу да користе и професори и ученици као приручник или збирку задатака за предмете чији програм покрива теме везане за објектно оријентисано програмирање и развој класа, као што су:

- Програмирање и програмски језици у трећем разреду Рачунарске гимназије,
- Програмирање и програмски језици у трећем разреду Математичке гимназије,
- Објектно оријентисано програмирање у трећем разреду гимназије за ученике са посебним способностима за рачунарство и информатику,
- Програмирање у трећем и четвртом разреду профила Електротехничар информационих технологија и Програмирање у четвртом разреду профила Електротехничар рачунара у електротехничким школама.

Све теме у књизи су обрађене кроз мноштво практичних примера креираних апликација чији је комплетан код изложен у тексту, уз потребна објашњења. Читаоцима се саветује да реализују ове решене примере на рачунару, а затим да приступе самосталном решавању нерешених задатака који су на крају сваког поглавља.

Осим примера у уводном поглављу, сваки пример приказује дефиницију неке класе, након чега следе примери једне апликације или више њих у којима се користе објекти написане класе. Приликом анализе појединачне дефиниције обавезно треба обратити пажњу на апликације које непосредно следе. Свака класа се пројектује на основу будуће употребе у различитим апликацијама. На

више места у књизи приказане су сличне апликације да би се видели и упоредили различити начини да се иста ствар програмерски реши.

Сви примери у књизи су урађени у програмском језику C#. За програмирање у програмском језику C# може да се користи бесплатно развојно окружење *Visual Studio Community:*

https://www.visualstudio.com/downloads/

https://www.visualstudio.com/vs/community/

1

Објектно оријентисано програмирање

Свака апликација са графичким корисничким интерфејсом коју познајемо као данашњи корисници рачунара састављена је од објеката. Објекти су: поље за унос текста, дугме, фудбалер или ауто у некој игрици.

Објекат обједињује у једну целину приватне податке и јавне методе (скуп правила о томе шта може да се ради са тим подацима и на који начин). Дефиницијом класе описани су сви објекти те класе.

Уколико је потребно да се креира апликација у којој се нешто исцртава, користиће се објекти неких од готових класа. Сваки објекат мора прво да се креира, а затим се позивају његове методе или се он користи у позивима метода. Следеће две команде креирају објекат класе *Graphics*, који садржи методе за цртање и оловку, објекат класе *Pen*, помоћу које ћемо цртати:

```
Graphics g = CreateGraphics();

Pen olovka = new Pen(Color.Red, 2);
```

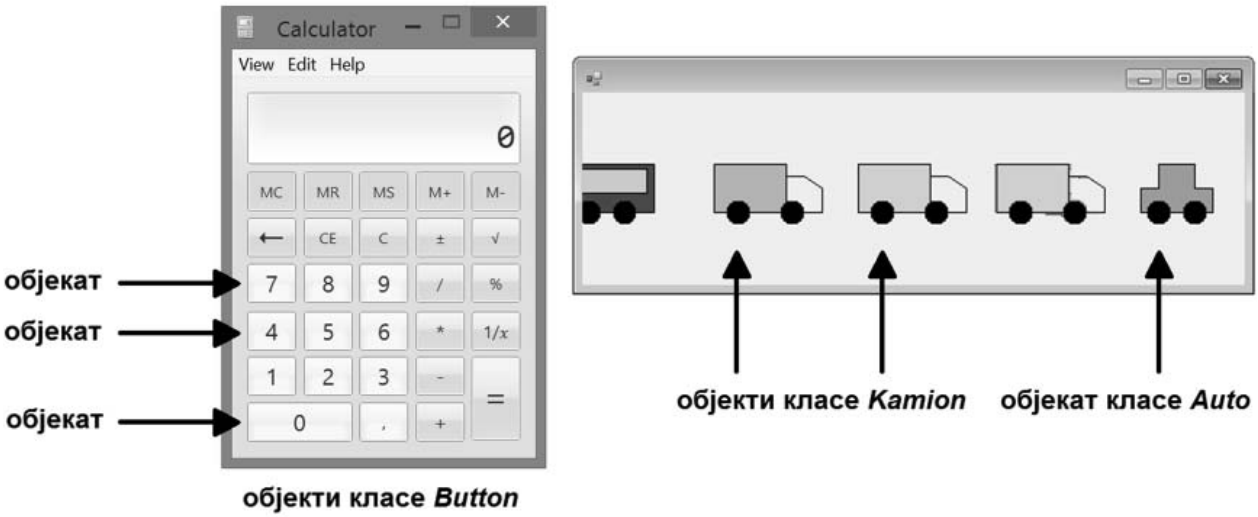
Линија од тачке (10, 20) до тачке (150, 210) ће се исцртати следећим позивом методе:

```
g.DrawLine(olovka, 10, 20, 150, 210);
```

Поглавље *Објектно оријентисано програмирање* садржи неколико једноставних примера помоћу којих ће бити објашњени основни концепти објектно оријентисаног програмирања и поступак креирања апликација употребом објеката готових класа.

Објекти и класе

Данашње апликације су креиране употребом објеката. Свако дугме, свако поље за унос текста, као и сваки други елемент корисничког интерфејса апликације јесте објекат. Заправо, и сама апликација је објекат.



Објекат обједињује у једну целину податке и функције (шта и на који начин може да се ради са тим подацима). Дефиницијом класе описани су сви објекти те класе. Опште правило је да су сви подаци приватни и на тај начин заштићени, док су све функције тј. методе јавне и помоћу њих се контролисано управља објектом. Ово правило се назива **енкапсулација**. Постоје ситуације у којима долази до одступања од овог правила, али су оне ретке. Приватни подаци се називају **приватни атрибути** класе, а јавне функције које су део класе називају се **јавне методе**.

Посматрајмо објекат *timer1* класе *Timer* (тајмер, часовник).

- Временски интервал је податак који нам говори о времену које протекне између два откуцаја тајмера (изражено у милисекундама):

```
timer1.Interval = 1000;
```

- Тајмер може да се покрене или заустави позивом метода (функција):

```
timer1.Start(); timer1.Stop();
```

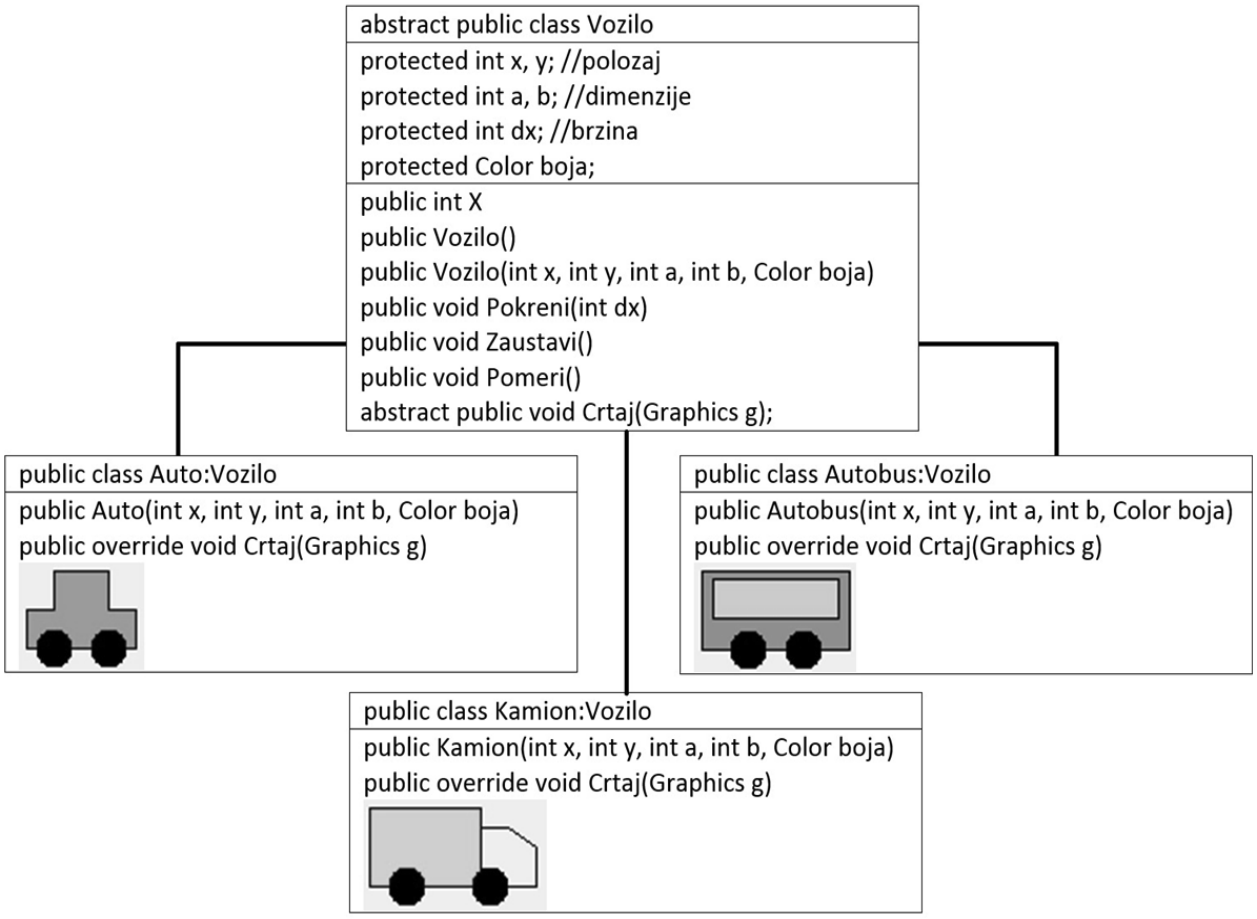
Сам податак, у случају тајмера, јесте приватан, али му приступамо помоћу **јавног својства** које у себи има уграђену контролу начина на који можемо са податком да поступамо. Дакле, иза јавног својства *Interval*, постоји приватни атрибут, којем не можемо директно да приступимо. Приватним атрибутима можемо да приступамо само унутар дефиниције класе, где у телу јавних метода дефинишемо шта са њим може да се ради, али изван класе, тј. у апликацијама у којима се објекти користе, можемо да користимо само јавне делове класе.

Да би се користио у апликацији, сваки објекат мора да се креира а то се ради употребом **конструктора**. Конструктор је посебна јавна метода која је такође део класе, која нема повратни тип и има идентичан назив као и класа. Неке класе, поред приватних атрибута, јавних метода и својстава јавног конструктора, могу да имају и друге елементе, као што су, на пример, дефиниције **оператора**.

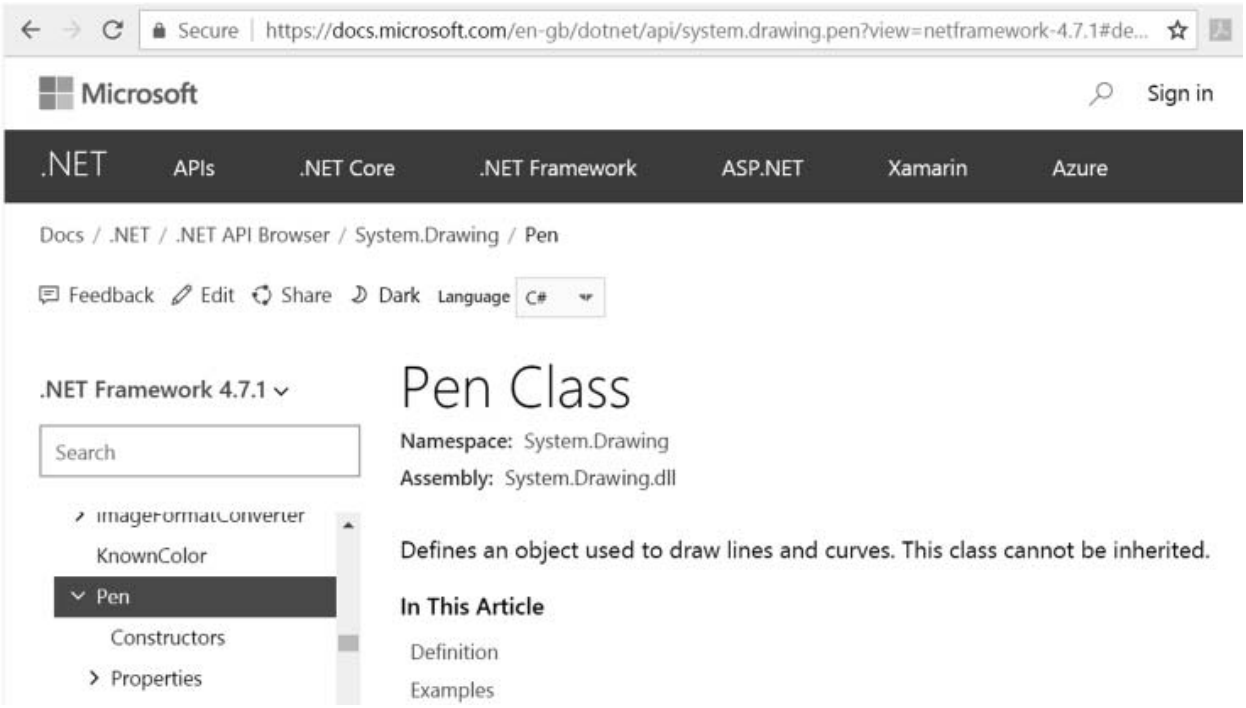
Споменутим јавним елементима дефиниције класе (конструкторима, својствима, операторима) биће посвећено више пажње у посебним поглављима.

Права предност објектно оријентисаног програмирања (скраћеница: ООП) види се тек када исту класу употребимо у неколико апликација – количина програмирања у свим следећим апликацијама где користимо објекте једном креиране класе далеко се смањује.

Механизам који такође олакшава програмирање и ствара мањи простор за грешке јесте **наслеђивање**. Све оно што је заједничко за неколико класа пише се само једном у натклаcи и тиме се избегава непотребно понављање. На пример, свако возило може да се покрене, креће, заустави, па ће се те функције издвојити у наткласу, док ће се изведене класе разликовати по приказу: ауто, аутобус и камион не изгледају исто.



Постоји велики број готових класа чије објекте можемо да користимо у апликацијама које креирамо. Те класе су груписане по **именским просторима** (енг. *namespace*), а комплетан списак може се пронаћи у онлајн документацији ***.NET API Browser***. За сваку класу у документацији може се пронаћи њен детаљан опис, као и примери употребе објеката у апликацијама.



Неке од готових класа чије ћемо објекте користити у апликацијама са графиком јесу: *Pen*, *Pens*, *SolidBrush* и *Graphics*. Ове класе су груписане у именски простор *System.Drawing* зато што све служе за цртање. Уколико, на пример, желимо да радимо са текстуалним фајловима, користићемо објекте класа *StreamReader* и *StreamWriter* из *System.IO* именског простора.

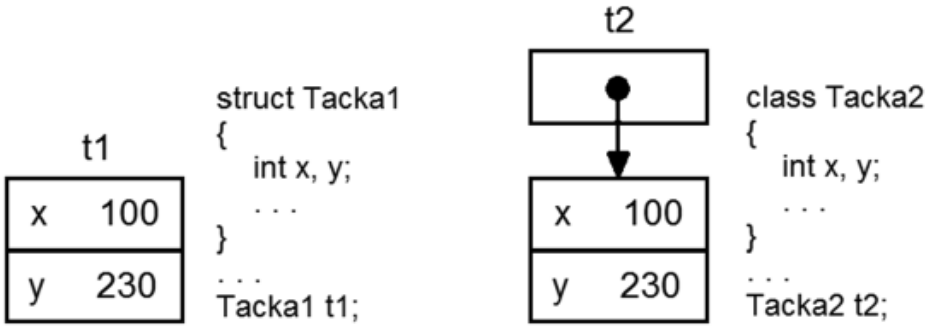
Програмски језик C# је креиран као чист објектно оријентисани програмски језик, што се види и у томе да су чак и основни типови података заправо сложени конструкти. За једну, на пример, целобројну променљиву можемо да позовемо методу за конверзију у текстуални податак, тј. радимо са њом као са објектом:

```
int a = 15;

textBox1.Text = a.ToString();
```

Основни типови података реализовани су као структуре. Тако је целобројни тип *int* заправо структура *Int32*. За разлику од класа, структуре спадају у вредносне типове. Класе спадају у референтне, што значи да уз сваки податак имамо и референцу, показивач.

Уколико бисмо имали задатак да дефинишемо тип података чији ће се објекти користити у апликацијама, можемо да изаберемо класу или структуру. Уколико одаберемо класу, уз сваки објекат ћемо у меморији имати и показивач на њега.



Најчешће се креирају класе, док је за неке једноставније ситуације згоднije употребити структуре. Тако су, од готових типова које користимо у апликацијама, *Point* и *Color* структуре.

Постоји много сличности у раду са објектима структура и класе, али постоје и разлике. У овој књизи се неће детаљно улазити у разлике, већ ће се највише причати о општим концептима рада са објектима, од којих се многи односе и на класе и на структуре, као и о принципима објектно оријентисаног програмирања.

Поред употребе готових класа, могуће је креирати разне нове класе и њихове објекте користити у апликацијама. Највећи део ове књиге биће посвећен управо томе.