- **Увод у развојно окружење програмског језика**

o*Основни елементи програмског језика C#*

Програмски језик C# је настао 2002. године у „радионици" чувене компаније Microsoft као производ жеље да се направи објектно оријентисани програмски језик (попут Јаве) који омогућава релативно лако креирање апликација у графичком корисничком окружењу (попут програмског језика Borland Delphi). Овај програмски језик је језик опште примене и намењен је изради апликација за *Microsoft .NET* платформу.

Синтаксне конструкције програмског језика C# су изграђене на уобичајеној азбуци (велика и мала слова абецеде, цифре, специјални знаци). Као и у сваком другом програмском језику постоје такозвене службене или резервисане речи које имају своје унапред дефинисано значење и не могу се користити за именовање било каквих података од стране програмера. Овде наводимо скуп резервисаних речи поређаних у лексикографском поретку:

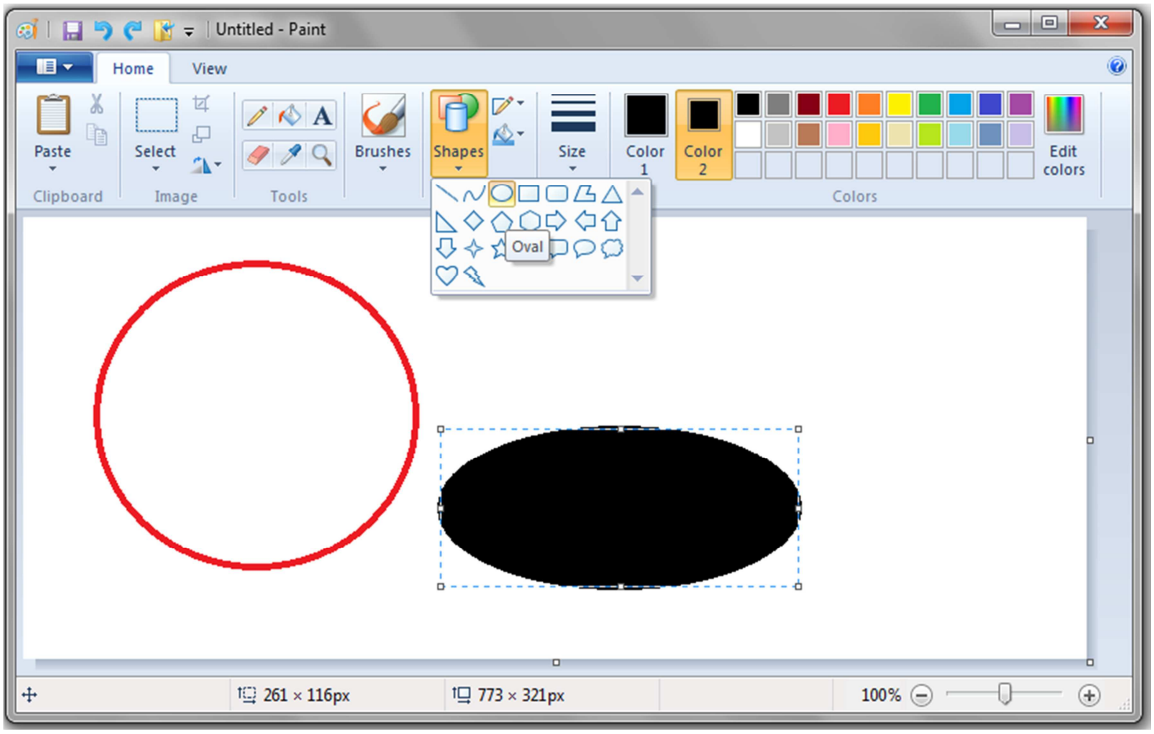
> abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default , delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object, operator, out , override, params, private, protected, public , readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, uint, ulong, unchecked, unsafe, ushort, using, virtual, volatile, void, while

У овом поглављу упознаћемо читаоца са поступком креирања апликација у овом окружењу. У каснијим поглављима ће бити више речи о типовима података којима C# манипулише као и о грађењу основних синтаксних и семантичких конструкција овог програмског језика.

o*Програми за цртање и класа Graphics*

Међу апликацијама заснованим на прозорима постоје разне апликације за цртање као и многе апликације, међу којима су најбројније игрице, које садрже много графичких елемената.

Погледајмо пример апликације *Paint* која је део *Windows Accessories* пакета. Корисник ове апликације може да црта по површини за цртање тако што контролише миша. Цртање започиње прикиском на тастер миша који активира догађај *MouseDown*, а завршава отпуштањем тастера, тј. догађајем *MouseUp*.



Можемо да мало детаљније анализирамо како ово изгледа са програмерске стране. Цртање по површини за цртање постижемо коришћењем метода уграђене класе *Graphics*. Објекат класе *Graphics* представља површину по којој цртамо. Прво га морамо креирати, а затим користити. Позивањем метода *CreateGraphics()* за неку контролу, креира се објекат класе *Graphics* тако да је површина за цртање управо та контрола.

Класа Graphics садржи методе за цртање разних облика, између осталог и:

- DrawLine – за цртање линија;
- DrawEllipse - за цртање елипсе;
- DrawRectangle - за цртање правоугаоника;
- FillEllipse - за цртање попуњене елипсе;
- FillRectangle - за цртање попуњеног правоугаоника.

Простор са цртање бришемо коришћењем метода *Clear* класе *Graphics*. Овај метод има један параметар, боју позадине (типа *Color*) контроле. Позивом метода *Clear* брише се простор за цртање и попуњава се бојом која је предата као параметар овом методу. На пример, позив метода

```
g.Clear(Color.White);
```

где је *g* објекат класе *Graphics*, попуњава контролу белом бојом.

Креирани објекат класе *Graphics* заузима одговарајуће ресурсе нашег система па је потребно, по завршетку цртања, ослободити ресурсе коришћењем метода *Dispose()* класе *Graphics*.

При коришћењу метода за цртање, неопходно је креирати оловку (објекат класе *Pen*) ако желимо да цртамо контуре жељеног облика,

или четку (објекат класе *SolidBrush*) ако желимо попуњен (филован) облик. Класа *Pen* је дефинисана у именском простору *System.Drawing*. Објектом класе *Pen* дефинишемо боју, ширину и стил линија и кривих које цртамо. Својство *Color* објекта класе *Pen* представља боју, док својство *Width* представља ширину линије.
Креирање објекта класе *Pen* постижемо позивом конструктора. Најчешће се позива конструктор коме предајемо, као параметре, редом, боју и ширину објекта. Без обзира на то који конструктор позивамо, његов први параметар је боја. Креирање објекта оловка класе *Pen* чија је боја црвена а ширина 5 постижемо на следећи начин:

Pen olovka=new Pen(Color.Red,5);

Када завршимо са коришћењем објекта класе *Pen*, потребно је извршити ослобађање ресурса које је тај објекат користио. То се постиже позивом метода *Dispose()*. Ослобађање ресурса које је заузео објекат оловка из претходног примера постижемо на следећи начин:

olovka.Dispose();

Класа *Pens* дефинисана у System.Drawing садржи својства за добијање оловке ширине један за све стандардне боје. Оловку црвене боје која представља својство класе *Pens* позивамо навођењем *Pens.Red.*
При цртању попуњених облика (правоугаоник, елипса) користимо објекте класе *SolidBrush* која је дефинисана у именском простору *System.Drawing*. Боја којом се врши попуњавање дефинише се својством *Color.* Креирање објекта класе *SolidBrush* остварујемо позивом конструктора те класе коме предајемо као параметар боју објекта. На пример, креирање објекта *cetka* класе *SolidBrush* извршавамо на следећи начин:

SolidBrush cetka=new SolidBrush(Color.Red);

Слично класи *Pens*, постоји класа *Brushes* која садржи својства за добијање четки за сваку стандардну боју.
Одмах по учитавању контроле, аутоматски се покреће догађај *Paint* којим дефинишемо изглед контроле. Овај догађај се такође покреће и када део контроле, који је претходно био преклопљен, поново постане видљив. Према томе, кад год се контрола исцртава, позива се метод за обраду догађаја *Paint*. Догађај *Paint* можемо позвати и експлицитно, у било ком делу апликације, позивом метода *Refresh()* за одговарајућу контролу.
Кад програмирамо догађај *Paint*, можемо користити објекат класе *Graphics* који је део прослеђеног параметра *e*, типа *PaintEventArgs (e.Graphics),* тог догађаја. Када користимо тај објекат, не позивамо метод *Dispose()* јер ми нисмо креирали објекат.

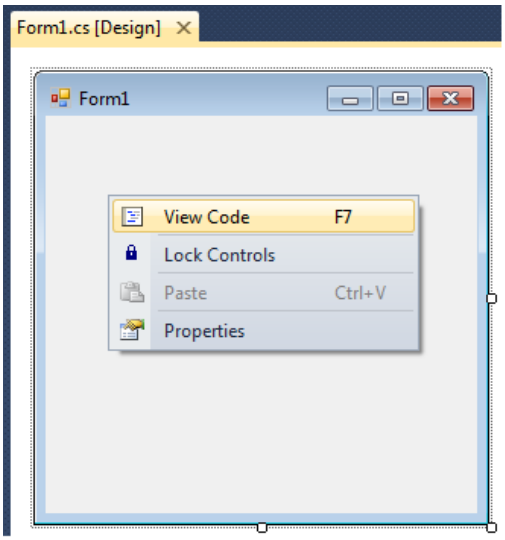
Свака површина по којој цртамо има сопствени координатни систем чији је почетак тачка (0,0), горњи леви угао те површине. Вредности X координата расту слева надесно, а вредности Y координата одозго надоле.

o***Развојно окружење и креирање апликације***

Можемо да креирамо апликацију у којој корисник црта пахуљице притиском на тастер миша. Након покретања *Microsoft Visual Studio C# (MSVC)* окружења, потребно је покренути нови пројекат. Процес креирања апликације је сложен и цео пројекат се састоји из много датотека (енг. files) које се чувају у једном фолдеру.

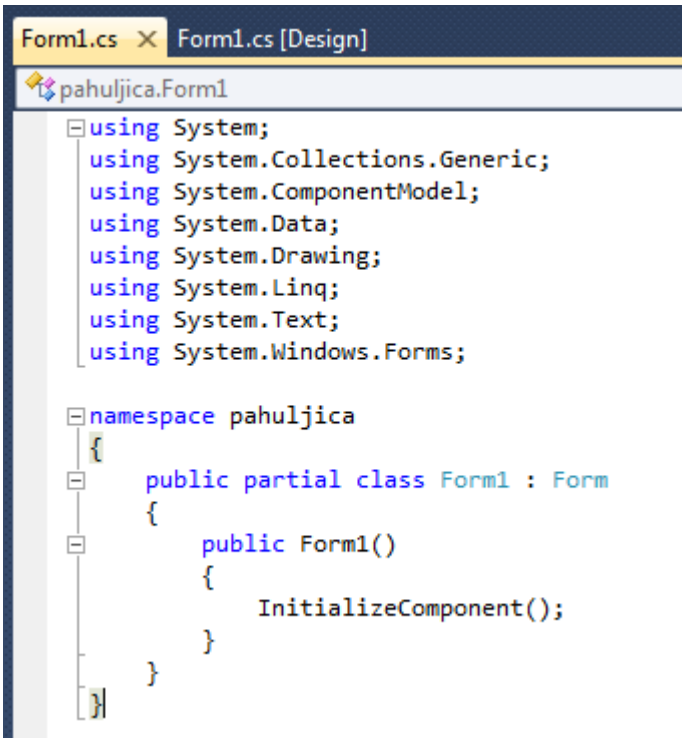
Тип пројекта који креирамо је *Windows Forms Application*. Све апликације које се користе у интегрисаним, графичким, системским окружењима данас су стандардизоване, па апликација у графичком корисничком окружењу представља скуп стандардних форми са којих се покрећу разни догађаји изазвани акцијама корисника или стањем система.

По покретању креирања нове апликације улазимо у режим дизајна стандардног *Windows* прозора, форме. Притиском на десни тастер миша и избором *View Code* можемо да видимо почетни код који генерише окружење.



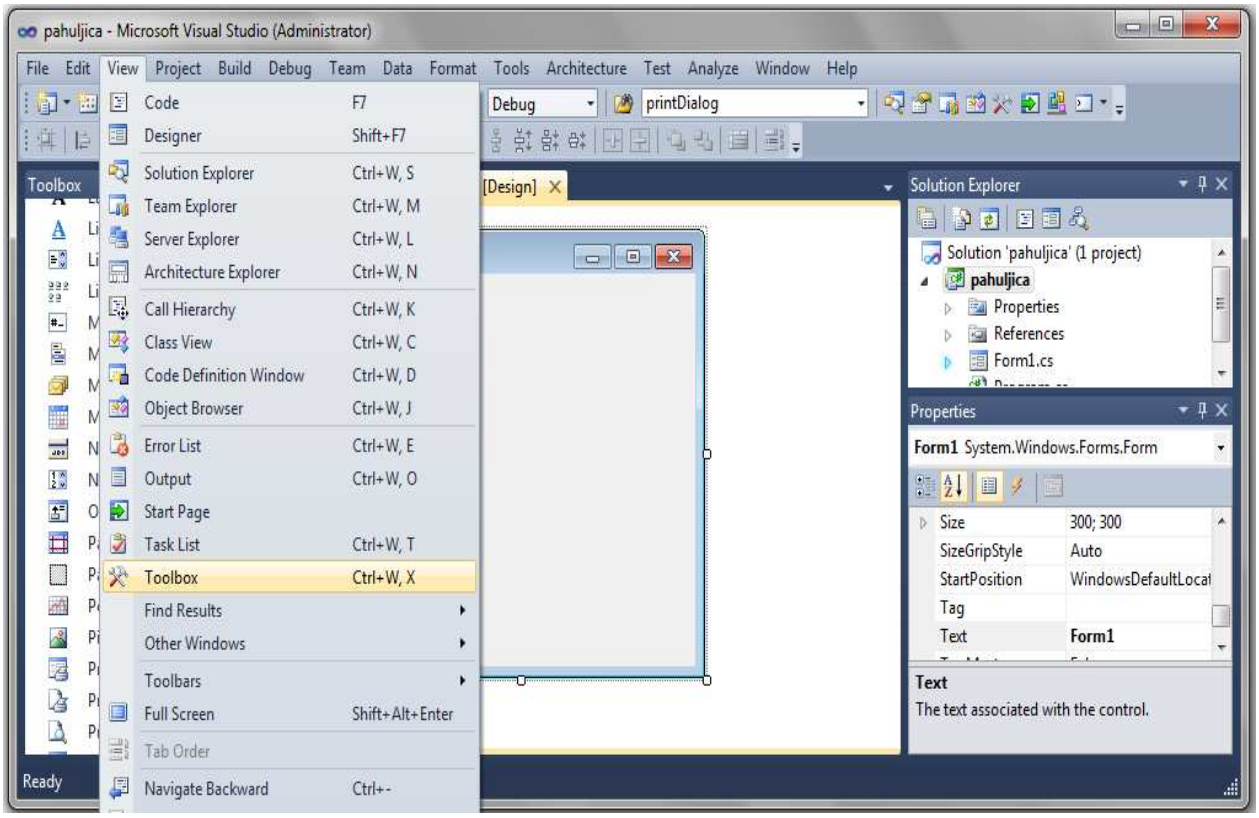
Стандардне форме, као и контроле које постављамо на њих, садржане су у системским библиотекама класа и доступне су свим програмерима који креирају апликације за одређени оперативни систем. Осим системских, програмери могу креирати и своје класе које касније користе у склопу различитих апликација. Класе се по различитим критеријумима које поставља програмер групишу у именске просторе (енг. *namespace*). У једном именском простору не могу постојати две класе истог имена а у различитим именским просторима могу.

При креирању апликације, C# аутоматски генерише нови именски простор. Уколико користимо неке класе које припадају другим именским просторима, неопходно је називе именских простора навести коришћењем директиве *using* (*using ImenskiProstor*).

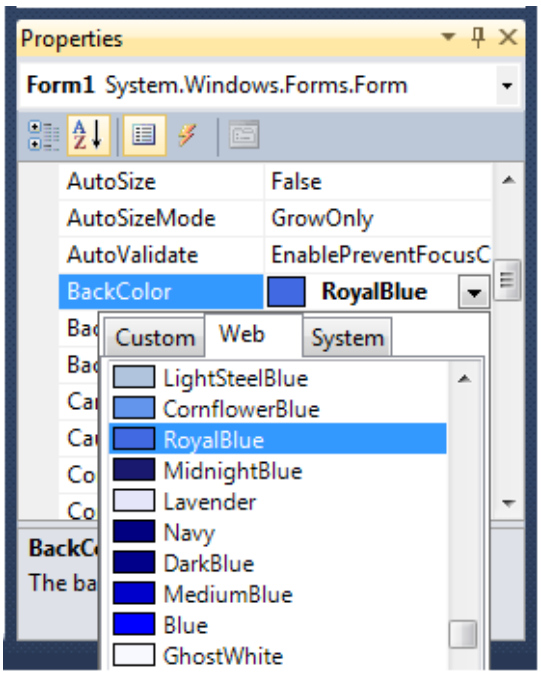


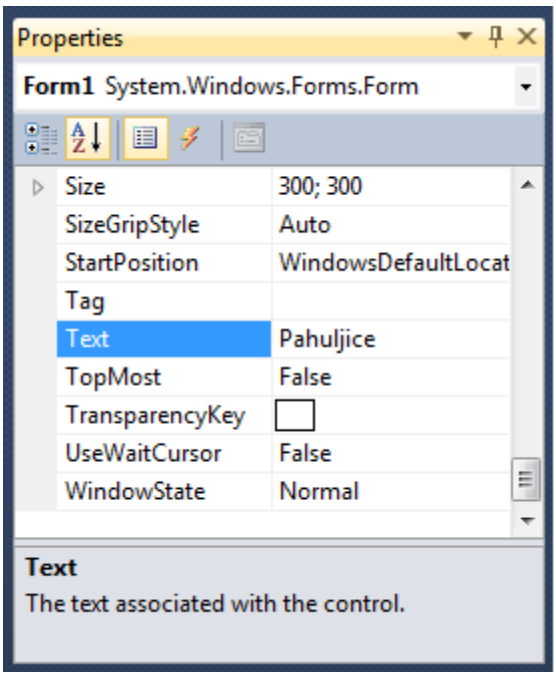
Системски именски простор који садржи све основне класе које користимо у апликацијама је *System.Windows.Forms*. Његове основне класе су *Control, CommonDialog* і *Form*. Класа коју програмер дефинише при креирању апликације наслеђена је од системске класе *Form*, што значи да садржи сва њена својства и методе. Програмер даље надограђује и мења класу. Графички, она представља прозор у коме се извршава апликација. Њој се могу додавати системске контроле које су наслеђене од класе *Control*.

Само развојно окружење нуди програмеру могућност да га прилагођава својим потребама тако што може да укључује или искључује разне помоћне прозоре и друге опције. Најчешће је ипак свима потребно да, док креирају апликацију, осим централног дела у којем граде саму апликацију виде и *Toolbox, Properties Window* и *Solution Explorer*. Уколико нису укључени, могу да се укључе кроз опције *View* падајућег менија.



Прозор *Solution Explorer* приказује све делове пројекта. У прозору *Toolbox* се налази списак контрола, објеката које можемо додавати у нашу апликацију. Прозор *Properties* nam omogućava rad sa objektima, i sadrži za svaku kontrolu spisak svih svojstava i događaja.
Можемо да променимо боју позадине форме у нашој апликацији тако што ћемо да променимо вредност својства *BackColor* на жељену боју у прозору *Properties*. Насловну линију форме можемо да дефинишемо вредношћу својства *Text*.





У прозору *Properties* можемо да мењамо и остала својства класе *Form*, а овде набрајамо нека од најчешће коришћених:

- Name

  име инстанце класе Form

- Text

  текст исписан у насловној линији форме

- Left i Top

  координате положаја левог горњег угла форме у активној резолуцији екрана

- BackColor

  боја позадине форме

- Width

  ширина форме

- Height

  висина форме

- Font

  врста слова коришћена заText

- FormBorderStyle

  дефинише изглед оквира (ако је подешено на Sizable, могућа је промена величине форме, док FixedDialog онемогућава промену величине форме)

- MaximumSize

  максимална дозвољена величина форме (максимална ширина и максимална висина; ако је подешена на 0, нема ограничења максималне величине)

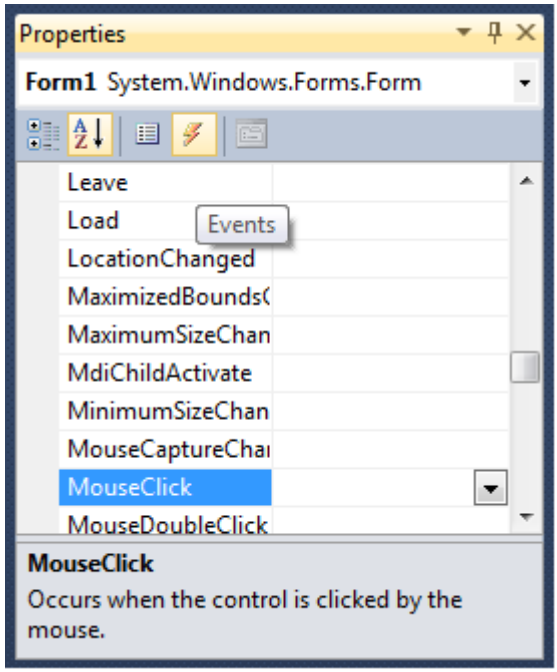
- MinimumSize

  минимална дозвољена величина форме (минимална ширина и минимална висина; ако је подешена на 0, нема ограничења минималне величине)

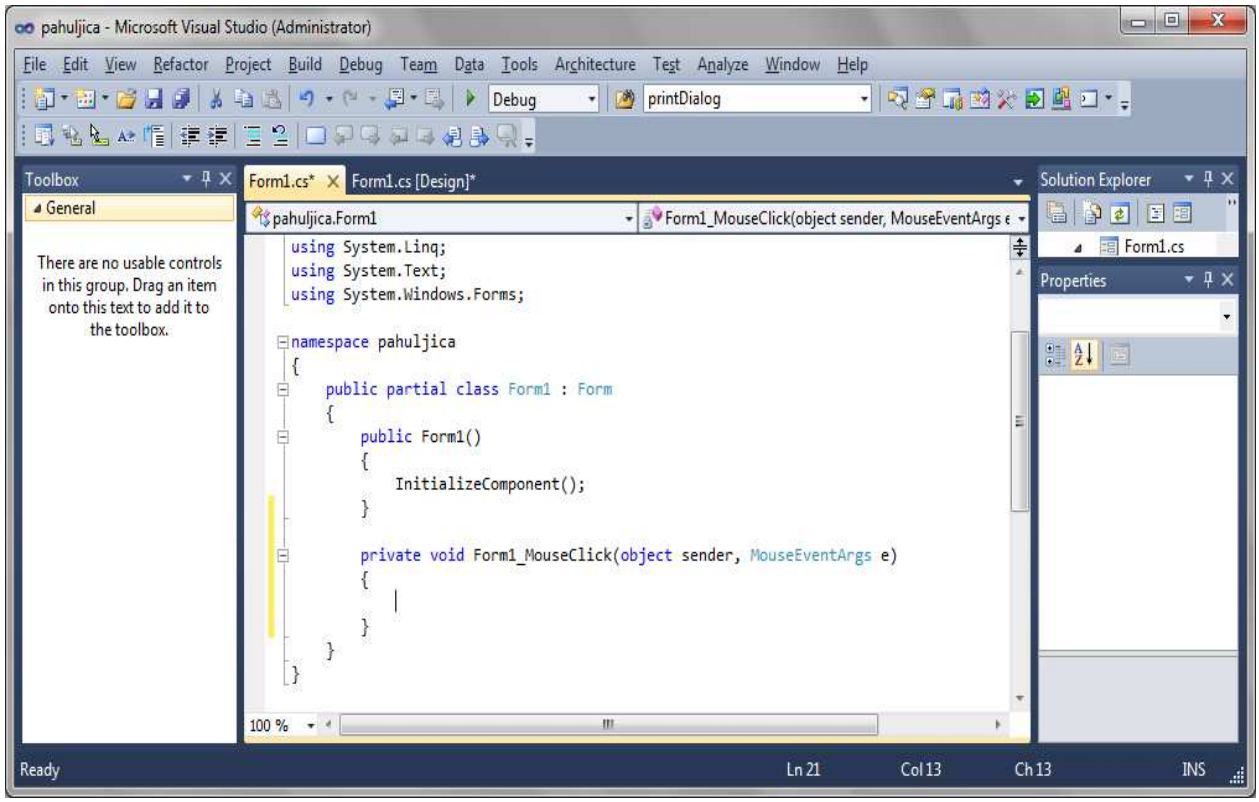
У току извршавања апликације, корисник изводи различите акције на форми. Неке од тих акција могу изазвати реакцију апликације ако је програмер испрограмирао одговарајући догађај. Такође, програмер може испрограмирати реакцију апликације на одређена стања система или на тренутно стање апликације.

У апликацији коју креирамо потребно је да се исцрта пахуљица на координатама на форми где је корисник притиснуо тастер миша. Притисак на тастер миша је догађај који можемо да пронађемо на списку догађаја у прозору *Propereties* испод знака за догађаје. Знак за догађаје је визуелно приказан као мања. Пронаћићемо догађај *MouseClick* и два пута ћемо да кликнемо на бело поље поред назива

догађаја да би окружење креирало простор у којем програмер уноси наредбе.

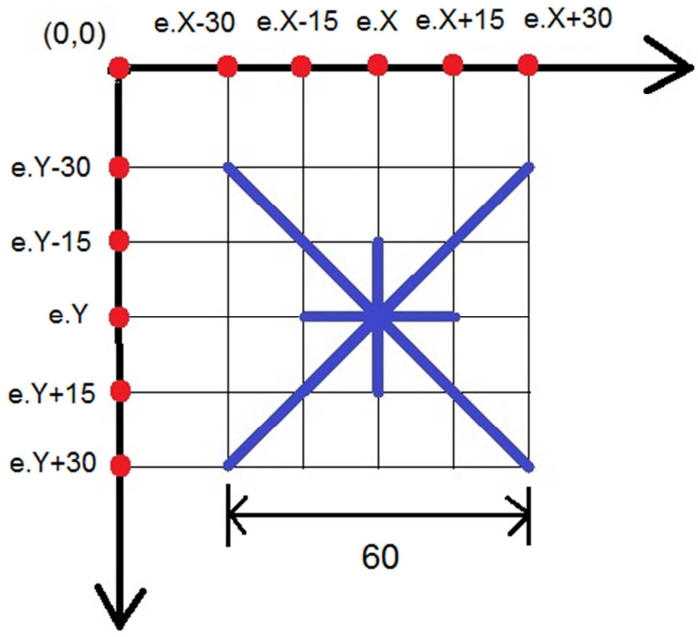


Појављује се простор у којем можемо да програмирамо као што је приказано на следећој слици.

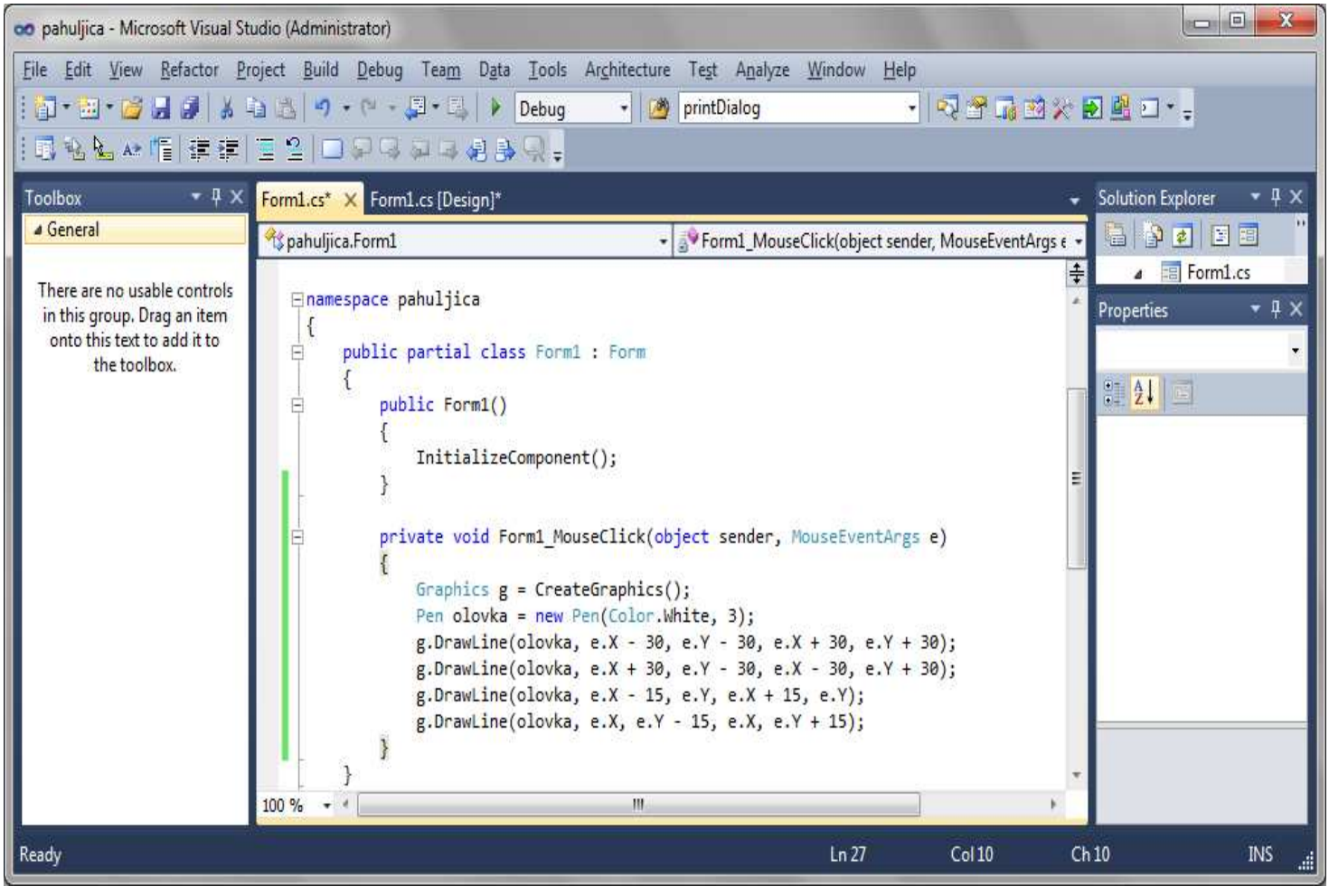


Пахуљицу цртамо тако што цртамо четири линије. Центар пахуљице је у тачки на коју је корисник притиснуо тастер миша. Та тачка је дефинисана вредношћу *X* и *Y* атрибута објекта *e* класе *MouseEventArgs* који дефинише параметре догађаја *MouseClick*.

На следећој слици можемо да анализирамо координате почетних и крајњих тачака четири дужи које ћемо да нацртамо. Величина пахуљице је фиксирана и износи 60 у висину и 60 у ширину. Координатни систем форме је тако оријентисан да је координатни почетак у горњем-левом углу, *x* координате расту на десно а *y* координате расту на доле.



Као што је већ објашњено, за цртање су нам потребне методе објекта класе *Graphics* као и објекат класе *Pen*, тј. оловка којом цртамо.



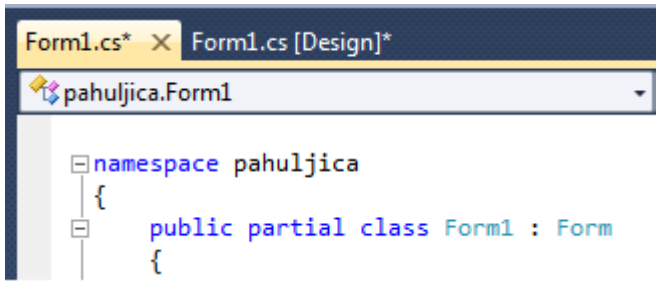
Набројаћемо још неке од метода које често програмирамо:

- Load – реакција на учитавање форме;
- Paint – реакција на појављивање (исцртавање) форме;
- MouseClick – реакција на клик дугметом миша;
- MouseDown - реакција на притисак на дугме миша;
- MouseUp - реакција на отпуштање дугмета миша;
- MouseMove – реакција на померање миша;
- KeyDown - реакција на притисак тастера тастатуре;
- KeyPress - реакција на откуцан тастер тастатуре;
- KeyUp - реакција на отпуштање тастера тастатуре;
- Resize – реакција на промену величине форме.

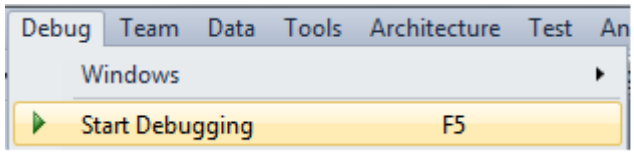
Можемо да уочимо да у изради апликације постоје две етапе:

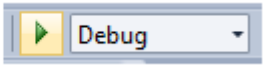
- етапа дизајна корисничког интерфејса;
- етапа кодирања.

Ове две етапе не морају бити независне, нити временски раздвојене већ се могу међусобно преплитати током израде апликације. У било ком тренутку можемо да се вратимо на кодирање избором језичка *Form1.cs* или на дизајн корисничког интерфејса избором језичка *Form1.cs[Design]*.

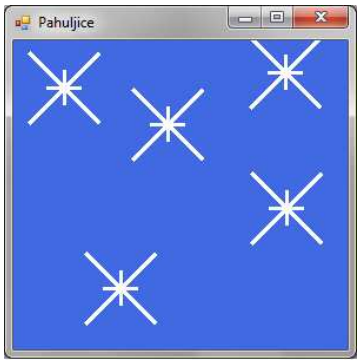


Апликацију коју смо креирали покрећемо на један од приказаних начина: избором ставке *Start Debugging* из *Debug* падајућег менија, кликом на зелену стрелицу или кликом на тастер F5.





Када се покрене апликација за цртање пахуљица, коју смо креирали, корисник може да црта беле пахуљице по плавој позадини форме.



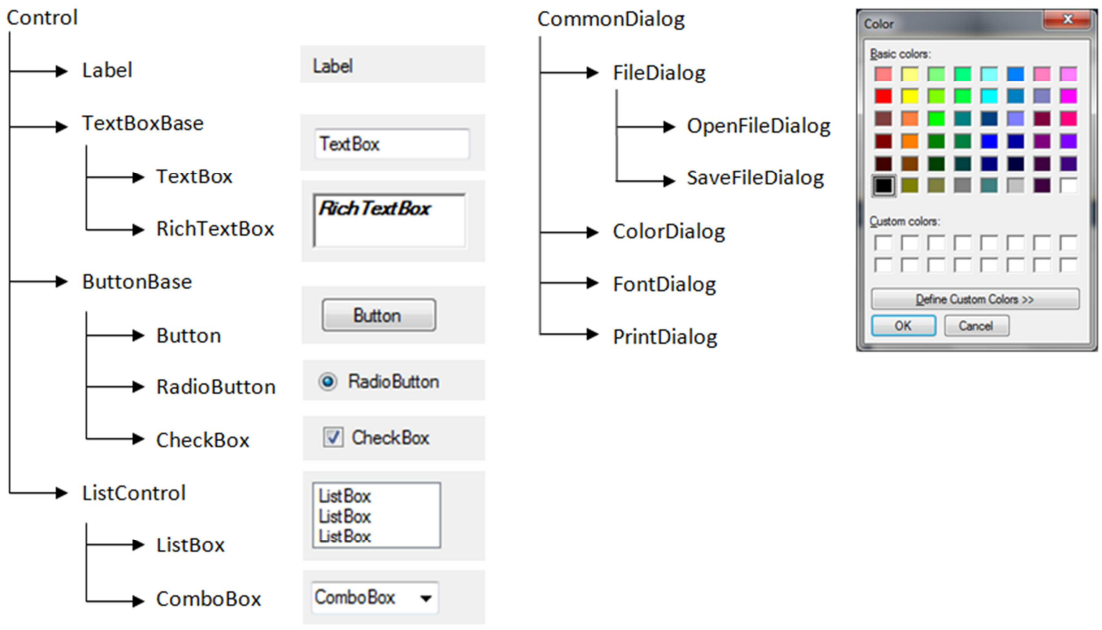
o***Креирање апликација и контроле***

Креирање апликација подразумева да програмер проширује форму додавањем нових, претходно дефинисаних компоненти, што системских, што кодираних. Затим, правилним избором догађаја које кодира, усмерава ток апликације. У класи, програмер може дефинисати потпуно нове методе али и предефинисати већ постојеће.

За разлику од првог примера који смо урадили, већина апликација има много сложенији изглед па су и етапа дизајна корисничког интерфејса и етапа кодирања много сложеније. Показаћемо то кроз још један пример. Креираћемо апликацију у којој ће корисник моћи сам да бира боју позадине, као и да намести да се боја аутоматски мења у правилним временским интервалима.

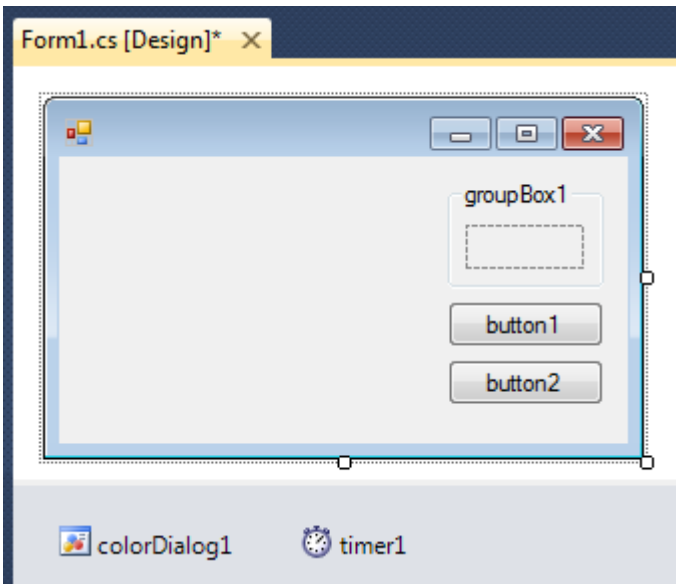
✓ ***Етапа дизајна корисничког интерфејса***

У оквиру ове етапе на форму, која представља основу за грађење апликације, постављамо потребне, претходно дефинисане компоненте. Претежно су то системски дефинисане контроле. Контроле су објекти помоћу којих корисник комуницира са апликацијом. На пример, корисник уноси податке у поље за унос текста и кликом на дугме покреће одређену акцију у апликацији. Такође, апликација резултате обраде података приказује кориснику кроз одговарајуће контроле. Већина контрола је изведена из класе *System.Windows.Forms.Control*. Због тога, многа својства и догађаји различитих контрола имају исто име и слично значење. На слици су наведене неке од најчешће коришћених класа изведених из системских класа *System.Windows.Forms.Control* и *System.Windows.Forms.Common-Dialog*.



Све компоненте које у фази дизајна можемо укључити у апликацију бирамо из палете алата (енг. *ToolBox*) развојног окружења MVSC.

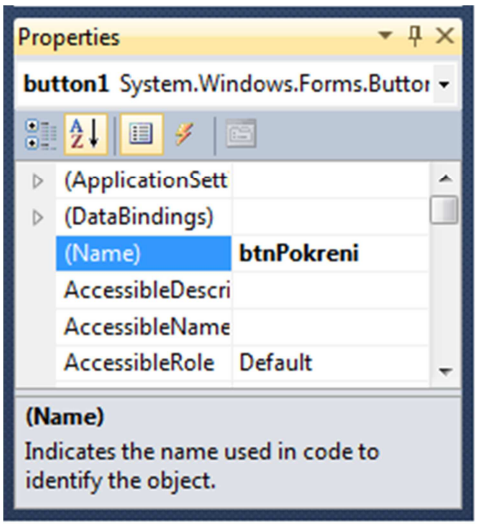
Покренимо нови пројекат имена *bojaPozadine*. Током етапе дизајна корисничког интерфејса додајемо два дугмета, једно поље за цртање (контрола *PictureBox*) унутар контроле *GroupBox*, дијалошки оквир *ColorDialog* и часовник *Timer*. Апликација треба да има изглед као на следећој слици.



Вредност својства *Name* је име које се при кодирању програма користи за приступ контроли. При постављању одређене контроле на форму, C# јој аутоматски додељује име које се састоји из имена класе (али почиње малим словом) и броја. Без обзира на то, било би добро да сваки програмер објекте именује по значењу и припадности класи. Када се на форму постави дугме, креира се један објекат класе *Button*.

Уколико је то дугме прво које смо поставили на форму, вредност својства *Name* је аутоматски *button1*. Програмер би могао да преименује објекат тако да прва слова представљају тип контроле (bt, btn) а затим следи реч која асоцира на употребу те контроле у програму. Вредност својства може се изменити или у прозору *Properties* интегрисаног окружења Visual Studio.NET, или инструкцијама програмског језика.

У нашој апликацији ћемо променити имена додатим контролама. Помоћу једног дугмета ће корисник моћи да покрене аутоматско мењање боје позадине у правилним временским интервалима, а другим ће то моћи да заустави. Према томе ћемо их именовати са: *btnPokreni* и *btnZaustavi*. Контроли *pictureBox1* ћемо дати име *pbBoja*.



Свака тачка (пиксел) на екрану има координате свог положаја. Почетак координатног система је тачка (0,0) која представља горњи леви угао екрана. Координатни систем има две осе. X оса је хоризонтална и вредност x координате расте надесно. Y оса је вертикална и вредност у координате расте надоле. Вредности координата су увек позитивни бројеви.

Свака контрола има свој локални координатни систем који је организован на исти начин, тј. координате горњег левог угла контроле су (0,0).

Контроле (дугме, лабела, поље за унос текста...) имају свој положај на форми на којој су постављене. Положај једне контроле се дефинише координатама њеног горњег левог темена у координатном систему форме. Својство *Top* представља удаљеност горње ивице контроле од врха, а *Left* удаљеност леве ивице контроле од леве ивице прозора.

Величина контроле је такође изражена бројем пиксела. Својство *Width* је ширина контроле и представља број пиксела по хоризонтали, од леве до десне ивице контроле, а *Height* је висина контроле.

Можемо подесити да се при промени величине прозора мења величина контроле. Коришћењем својства контроле *Anchor*, дефинишемо за које ивице прозора (Top, Bottom, Left, Right) је

„причвршћена" та контрола. Растојање између контроле и ивице прозора за коју је контрола „причвршћена" је константно. На пример, ако смо контролу причврстили за леву и десну ивицу прозора, при промени ширине прозора долази до промене ширине контроле.

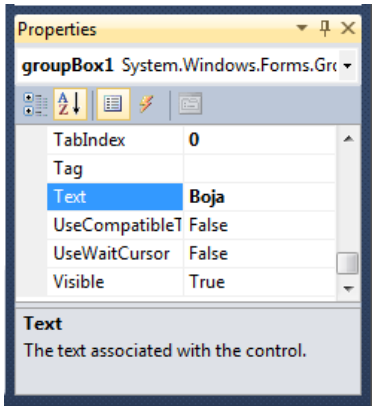
Вредност својства *Enabled* може бити *true* или *false*. Уколико је вредност *true*, корисник може приступити контроли (на пример, корисник може да кликне на дугме или да унесе текст у поље за унос текста), у супротном не може.

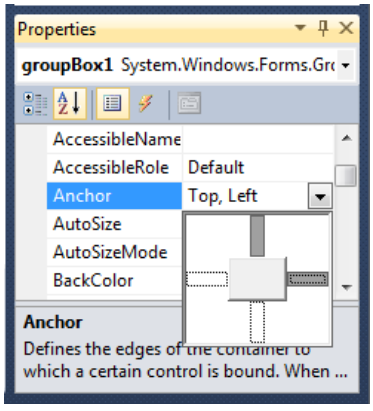
Својство *Visible* такође има две могуће вредности, *true* и *false*, којима је одређена видљивост контроле при извршавању апликације.

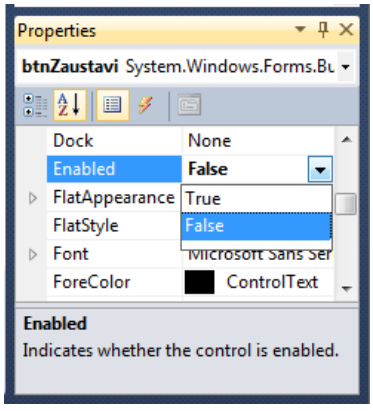
Боја предњег плана контроле, тј. боја слова на контроли и боја позадине контроле могу се мењати променом вредности својстава *ForeColor* и *BackColor.*

Свакој контроли је придружен неки текст који је описан својством *Text*, а врста слова којом је исписан тај текст описана је својством *Font*.

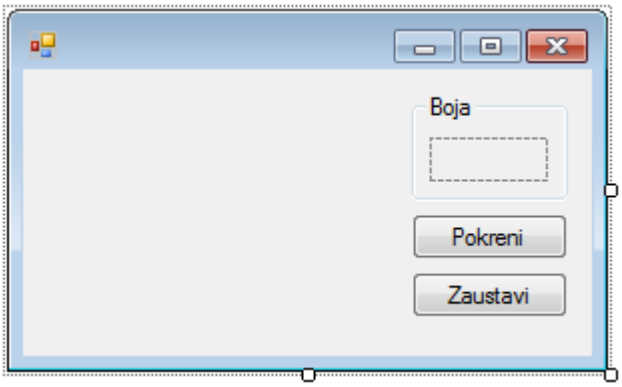
У нашој апликацији ћемо променити вредност својства *Text* свим додатим контролама. Поред тога, нема смисла да зауставимо мењање боје уколико га нисмо покренули, тако да вредност својства *Enabled* за дугме *btnZaustavi* треба у почетку да буде *false*. Све контроле ћемо да вежемо за горњу и десну ивицу форме помоћу вредности својства *Anchor*.







Етапа дизајна наше апликације је у овом моменту завршена и апликација изгледа као на следећој слици.



Набројаћемо, уз објашњења, неке од најчешће коришћених контрола изведених из *System.Windows.Forms.Control*:

- Button

  је контрола која најчешће обезбеђује покретање одређених акција;

- RadioButton

  је контрола избора (од више RadioButtona у групи, у једном моменту може бити означен највише један);

- CheckBox

  је контрола потврде која може бити означена или не (употребом више CheckBoxova можемо омогућити избор прозвољног броја наведених опција);

- TextBox

  је контрола која представља поље за унос текста; карактеристичан је јединствен формат за сваки знак у тексту;

- RichTextBox

  је контрола која представља поље за унос форматираног текста; могуће је променити формат сваком појединачном знаку у тексту;

  Садржај ове две контроле дефинисан је својством Text типа string, чија је максимална дужина дефинисана својством MaxLength. Својство MultiLine (true ili false) одређује могућност писања текста у више линија, а својство ReadOnly одређује да ли се може писати у поље (false) или не (true). При раду са означеним делом текста, користимо својства SelectedText (текст који је означен), SelectionLength (број карактера у означеном делу текста) и SelectionStart (почетак означеног текста). Својством WordWrap дефинишемо да ли у контроли за текст са више редова реч аутоматски прелази у нови ред (true) када текст премаши ширину контроле или не (false).

- Label

  је контрола која служи за приказ текста и често се на форму поставља уз друге контроле да би означила њихову намену; постоје две врсте ове контроле:

  - стандардна Label
  - за повезивање са другим објектима, LinkLabel (hiperlink);

- ListBox

  је контрола која омогућава приказ листе елемената са које корисник може изабрати једну или више ставки (дефинишемо коришћењем својства SelectedMode);

- ComboBox

  је контрола која у себи обједињује неке могућности контрола TextBox и ListBox; попут ListBoxa садржи листу ставки од којих се може изабрати само једна; омогућава уношење текста као и TextBox;

- GroupBox

  је контрола која омогућава груписање више контрола у једну целину; најчешће је користимо у комбинацији са RadioButton-ом; у оквиру једног GroupBoxa може бити означен највише један RadioButton; често користимо GroupBox за обједињавање контрола које истовремено треба омогућити или онемогућити (Enabled), односно које истовремено треба приказати или сакрити на форми (Visible);

- PictureBox

  је контрола коју користимо за приказивање слике (својство Image).

✓ ***Етапа кодирања***

У овој етапи програмирамо методе који представљају реакцију апликације на одређени догађај изазван акцијом корисника или стањем система. Програмирањем ових метода усмеравамо ток апликације и омогућавамо кориснику безбедан и удобан рад.

Креирање апликације подразумева превасходно програмирање метода који се активирају при одређеном догађају. Најчешће су то догађаји који настају као последица одређене акције корисника (клик на дугме, промена садржаја поља за текст, избор неке од опција…), али могу се програмирати и методи везани за догађаје које не контролише корисник, већ, на пример, систем (истек одређеног временског интервала, учитавање објекта, попуњавање меморијских ресурса…). У сваком случају, сваки догађај је потпуно одређен оним ко га је изазвао (*sender*), као и одређеним параметрима који га описују (*EventArgs*). У зависности од догађаја, параметри који га описују се разликују. Тако је за догађај *MouseClick* дефинисан положај стрелице миша у тренутку дешавања (*e.X*, *e.Y*), док је, рецимо, за догађај *Paint* важан параметар *e.Graphics*.

Методи које програмирамо представљају заокружене и донекле независне програмске целине које се могу користити у различитим ситуацијама и које реагују различито у различитим околностима. Инструкције, из којих је метод састављен, извршавају се редом којим су наведене, једна за другом. По свом значењу и по сложености инструкције можемо поделити на

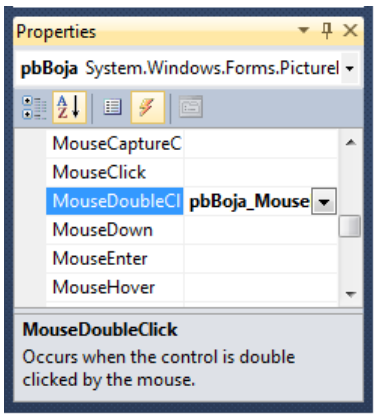
- декларације / дефиниције променљивих (објеката);
- инструкције доделе;
- инструкције гранања;
- инструкције циклуса.

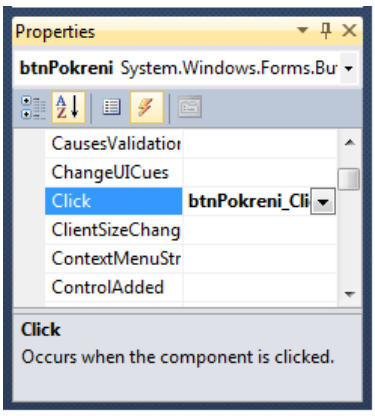
Уколико у методу користимо само прве две врсте инструкција, можемо рећи да је тај метод линијске структуре, што значи да се свака инструкција наведена у њему извршава тачно једном. По својој структури такви методи су најједноставнији и њих није могуће користити за решавање сложених проблема на које у програмирању наилазимо.

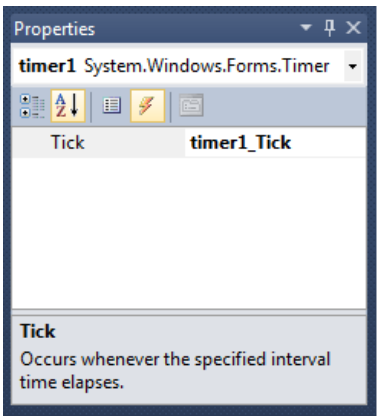
Поред инструкција можемо да пишемо и коментаре који доприносе читкости кода. У програмском језику C#, по навођењу "//", пишемо коментаре. Почев од "//" па до краја линије написани коментар се при превођењу програма игнорише. Уколико пишемо опширнији коментар, који обухвата више редова, од осталог дела кода одвајамо га на почетку са "/*" а на крају са "*/".

У нашем примеру ћемо програмирати методе линијске структуре које представљају реакције на следеће догађаје. Двоструким кликом мишем на контролу *pbBoja*, кориснику треба да се прикаже дијалошки оквир *colorDialog1* помоћу којег он бира нову боју позадине. Након приказа дијалошког оквира промениће се боја позадине форме и боја позадине контроле *pbBoja* на изабрану боју. Чак и уколико се касније боја позадине форме буде мењала аутоматски, боја позадине контроле *pbBoja* ће задржати последњу боју коју је корисник сам изабрао. Када корисник притисне дугме *btnPokreni*, потребно је да се активира часовник који ће у правилним временским интервалима мењати боју позадине форме. Када је покренута аутоматска промена боја, дугме *btnPokreni* треба да постане неактивно, а дугме *btnZaustavi* активно да би могли процес у жељеном тренутку да зауставимо.

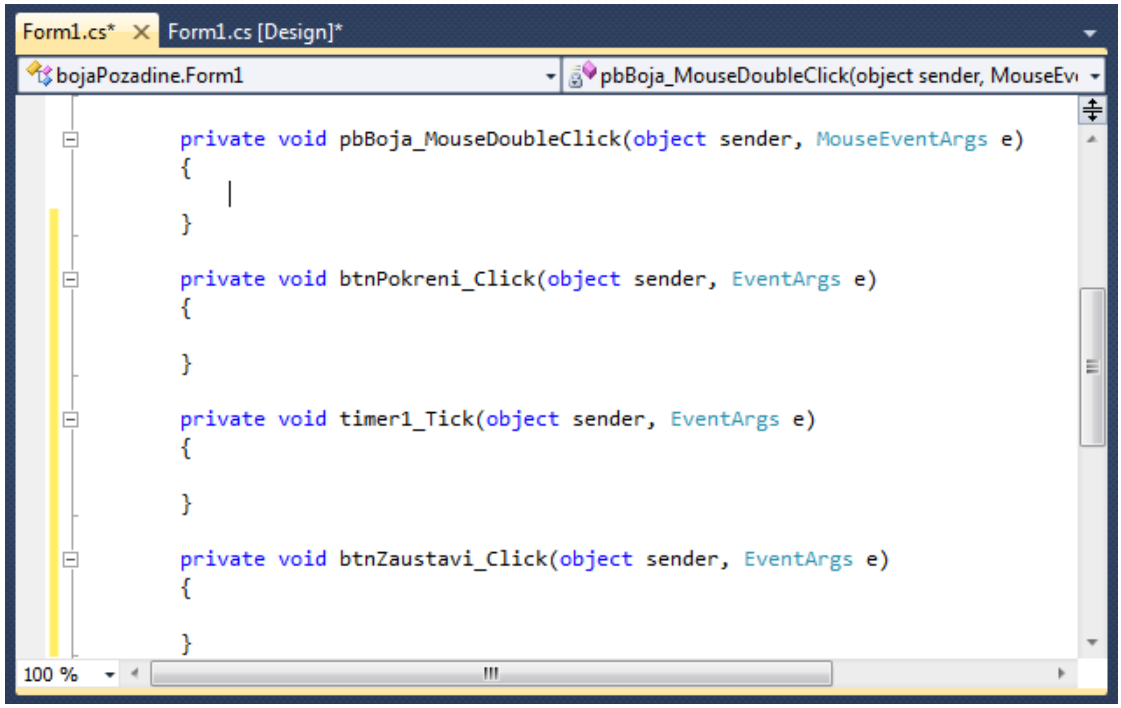
Потребно је да у прозору *Properties* изаберемо догађаје за сваку појединачну контролу са којом ће корисник имати интеракцију током рада апликације. То су *MouseDoubleClick* за контролу *bpBoja*, *Tick* за часовник *timer1*, *Click* за дугмад *btnPokreni* и *btnZaustavi.*



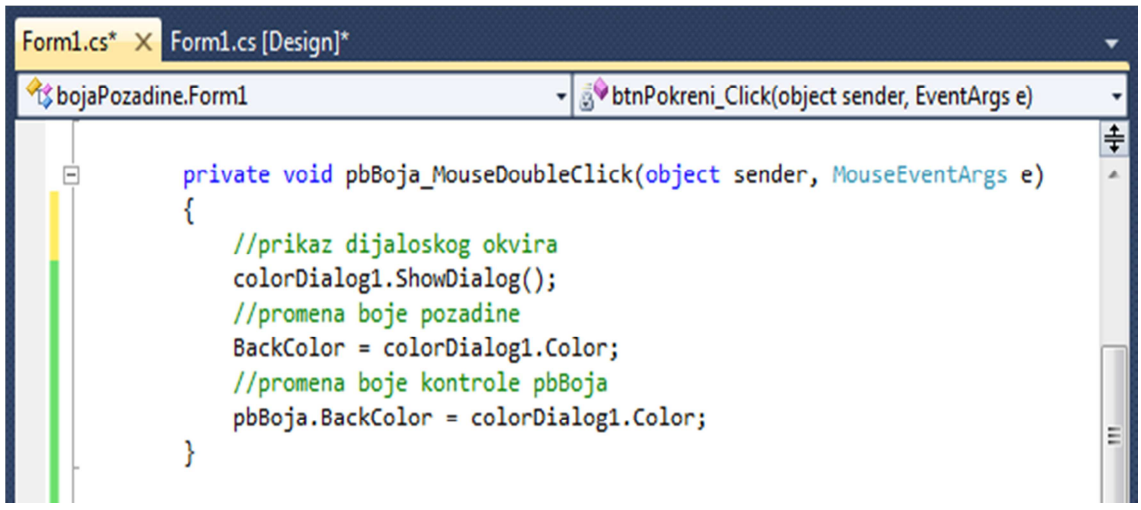




Окружење генерише оквир за писање метода које представљају реакције на изабране догађаје као што се види на следећој слици.



У оквиру реакције на догађај *MouseDoubleClick* за контролу *pbBoja* важно је да прикажемо дијалошки оквир *colorDialog1*. Основни метод свих класа које наслеђују класу *CommonDialog* је *ShowDialog()* којим се постиже приказивање дијалога. Дијалошки оквир *ColorDialog* има својство *Color* чија вредност представља изабрану боју. Промену боје остварујемо променом вредности својства *BackColor.* Резултат програмирања реакције на први од свих догађаја у овој апликацији изгледа овако:



Када корисник притисне дугме *btnPokreni*, потребно је покренути часовник и активирати дугме *btnZaustavi*. Притисак на дугме *btnZaustavi* треба да заустави часовник. Сваким откуцајем (енг. Tick) часовника је потребно да се на насумичан начин промени боја позадине форме.

Класу *Random* користимо за генерисање случајних бројева. Прво креирамо објекат те класе позивом конструктора класе:

Random R = new Random();

Метод којим генеришемо случајне целе бројеве је *Next*и он има три варијанте:

- *Next(maxVrednost)* враћа ненегативан случајан 32-битни цео број типа int, мањи од *maxVrednost*;

- *Next(minVrednost, maxVrednost)* враћа случајан 32-битни цео број типа *int*, већи или једнак *minVrednost* и мањи од *maxVrednost*.

За регистровање боје користи се структура *Color*. Ова структура садржи атрибуте којима су представљене најчешће коришћене боје (Red, Green, Blue, Yelow, Black, White...) али и метод којим можемо дефинисати произвољну боју:

FromArgb(nivoCrvene, nivoZelene, nivoPlave).

Вредности *nivoCrvene, nivoZelene, nivoPlave* су цели бројеви од 0 до 255. Случајну боју можемо дефинисати случајним избором ових вредности.

Етапу кодирања нашег примера завршавамо исписивањем следећег кода:

```
private void btnPokreni_Click(object sender, EventArgs e)
{
   timer1.Interval = 500;
   timer1.Start();
   btnPokreni.Enabled = false;
   btnZaustavi.Enabled = true;
}

Random r = new Random();

private void timer1_Tick(object sender, EventArgs e)
{
   BackColor = Color.FromArgb(r.Next(256), r.Next(256), r.Next(256));
}

private void btnZaustavi_Click(object sender, EventArgs e)
{
   timer1.Stop();
   btnPokreni.Enabled = true;
   btnZaustavi.Enabled = false;
}
```

Набројаћемо још неке догађаје везане за контроле изведене из класе *Control* за које програмирамо методе који представљају реакцију на њих:

- Click

  притисак тастера Ентер када је контрола активна или клик дугметом миша на контролу;

- KeyDown

  притисак неког од тастера са тастатуре док је контрола активна; овај догађај увек претходи догађајима *KeyPress* и *KeyUp* и преноси код тастера (*int*) који је притиснут;

- KeyPress

  притисак неког од тастера са тастатуре док је контрола активна; овај догађај се дешава увек после догађаја *KeyDown*, а пре догађаја *KeyUp* и преноси вредност типа *char* тастера који је притиснут;

- KeyUp

  отпуштање тастера тастатуре док је контрола активна; овај догађај преноси код тастера (*int*) који је притиснут;

- MouseClick

  клик дугметом миша на контролу;

- MouseDown

  притисак дугмета миша на контроли;

- MouseUp

  отпуштање притиснутог дугмета миша;

- MouseMove

  континуирано кретање миша преко контроле;

Догађаји који реагују на употребу тастера миша (*MouseClick*, *MouseDown*, *MouseUp*, *MouseMove*) преносе позицију (X,Y) показивача миша на контроли, као и информацију о дугмету које је притиснуто (лево или десно).

- Paint

  исцртавање контроле; овај догађај се покреће аутоматски по учитавању контроле, али и сваки пут када је потребно изнова исцртати контролу; догађај може позвати и програмер експлицитно у било ком делу апликације позивом метода Refresh() за одговарајућу контролу.

✓ *Задаци*

1. **Креирати апликацију којом се покреће исцртавање прскалица на позицији одређеној положајем стрелице миша у тренутку притиска дугмета миша. Свака прскалица треба да буде у нијансама црвене боје. Цртање престаје када отпустимо тастер миша.**

   Као што је већ раније објашњено, за регистровање боје користи се структура *Цолор* која има метод којим можемо дефинисати произвољну боју:

   FromArgb(nivoCrvene, nivoZelene, nivoPlave).

   Вредности nivoCrvene, nivoZelene, nivoPlave су цели бројеви од 0 до 255. Уколико желимо црвену боју, nivoCrvene би морао да има максималну док би nivoZelene и nivoPlave морали да имају минималну вредност па би метод позвали на следећи начин:

   FromArgb(255, 0, 0).

Следећа боја би била нијанса црвене јер има велики ниво црвене, а мале нивое зелене и плаве:

FromArgb(198, 25, 51).

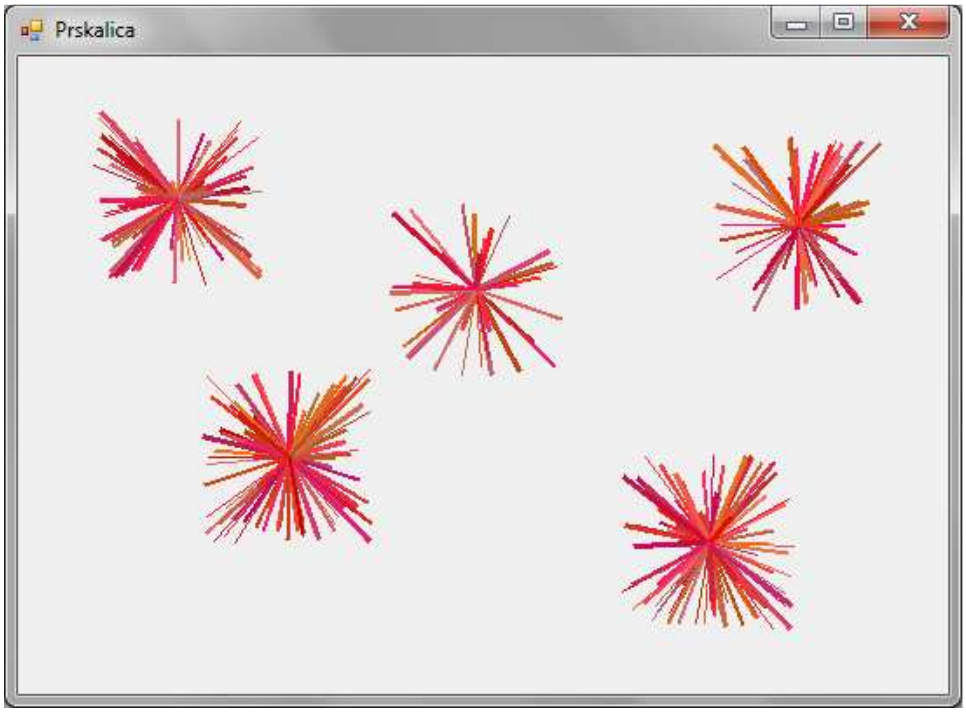
Нијансе црвене боје ћемо да добијемо тако што ћемо да контролишемо објекат класе *Random* и захтеваћемо да нам методом *Next* генерише већу вредност (између 180 и 255) за црвени ниво, а малу (испод 100) за зелени и плави.

За представљање једне тачке користимо две целобројне координате, *x* и *y* координату тачке.

Као што смо раније напоменули, линије цртамо коришћењем метода *DrawLine* класе *Graphics*. Овај метод можемо позвати на више начина, али најчешће користимо цртање линије која повезује тачке (x1,y1) и (x2,y2):

DrawLine(Pen olovka, int x1, int y1, int x2, int y2)

Параметар *оловка* је објекат класе *Пен* којим дефинишемо боју, ширину и стил линије коју цртамо.



Slika 33

```
int X, Y;
Random r = new Random();

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
   X = e.X;
    Y = e.Y;
   timer1.Start();
   timer1.Interval = 30;
}

private void timer1_Tick(object sender, EventArgs e)
{
   int x1= r.Next(X - 50, X + 50);
```

```
        int y1 = r.Next(Y - 50, Y + 50));
        Graphics g = CreateGraphics();
        Color boja=Color.FromArgb(r.Next(180,256), r.Next(100), r.Next(100));
        Pen olovka=new Pen(boja, r.Next(1,4));
        g.DrawLine(olovka, X, Y, x1, y1);
    }

private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        timer1.Stop();
    }
```

2. **Креирати апликацију којом се цртају линије којима је координата почетне тачке одређена положајем стрелице миша у тренутку притиска дугмета миша, а координата крајње тачке одређена положајем стрелице миша у тренутку отпуштања дугмета.**

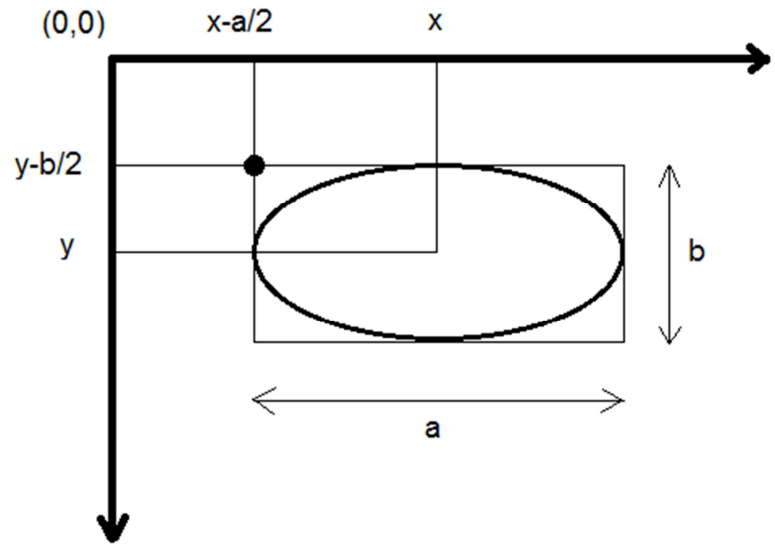
Како линију можемо нацртати само када су одређене координате и почетне и крајње тачке, метод *DrawLine* позивамо по отпуштању дугмета миша, дакле у догађају *MouseUp*. Координате почетне тачке одређујемо у тренутку притиска дугмета, у догађају *MouseDown*. Параметар ових догађаја, *MouseEventArgs e*, садржи и информацију (*e.X, e.Y*) о локацији стрелице миша при извршавању догађаја.

```
int x, y;
private void Form1_MouseDown(object sender, MouseEventArgs e)
{ //postavljanje koordinata pocetne tacke na trenutnu poziciju misa
    x = e.X;
    y = e.Y;
}
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    Graphics g = CreateGraphics();
    Random R = new Random();
    Pen olovka = new Pen(Color.FromArgb(R.Next(256),
            R.Next(256), R.Next(256)));
    //crtanje linije od pocetne do krajnje tacke, (e.X, e.Y)
    g.DrawLine(olovka, x, y, e.X, e.Y);
}
```

3. **Креирати апликацију у којој корисник црта кругове. Притиском на тастер миша почиње цртање круга са центром у одабраној тачки. У правилним временским интервалима повећава се полупречник круга док не отпустимо тастер миша. Омогућити избор боје.**

У горњем делу форме је постављен и објекат *pictureBox1* који у сваком тренутку приказује изабрану боју за цртање, односно бојење кругова. Боја се мења кликом на *pictureBox1*.

Круг је специјални случај елипсе па за његово цртање користимо метод *DrawEllipse*. Метод за цртање елипсе захтева да се прецизирају координате горњег левог темена правоугаоника описаног око елипсе, као и његове димензије, ширина и висина.



Slika 1

На следећи начин можемо да исцртамо елипсу са слике:

g.DrawEllipse(olovka, x – a/2, y – b/2, a, b);

Код круга су ширина и висина једнаке и износе двоструку вредност полупречника. Уколико је полупречник круга *r*, круг са центром у тачки (x, y) можемо да нацртамо на следећи начин:

*g.DrawEllipse(olovka, x – r, y – r, 2*r, 2*r);*

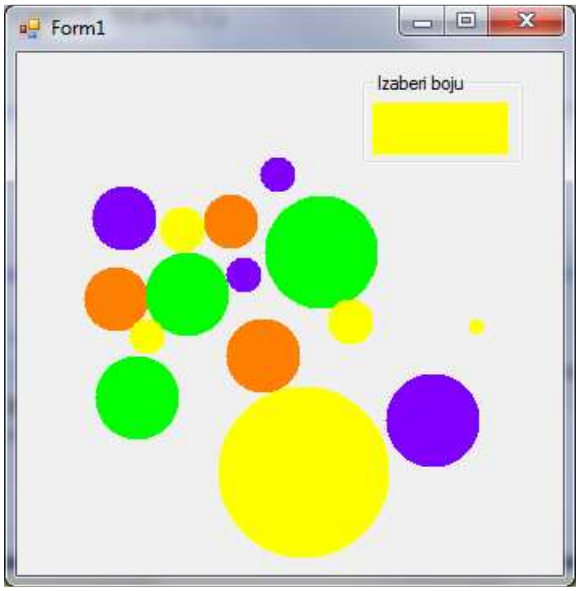
Слично, ако желимо да нацртамо круг попуњен неком бојом, користимо метод *FillEllipse:*

*g.FillEllipse(cetka, x – r, y – r, 2*r, 2*r);*

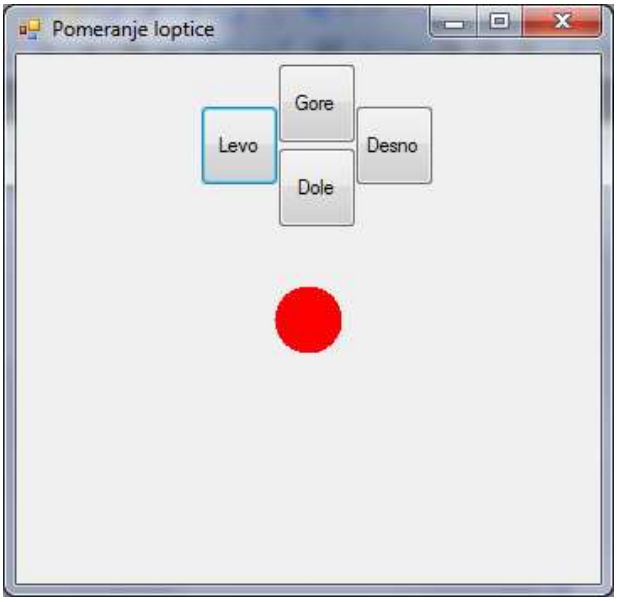
```
private void pictureBox1_Click(object sender, EventArgs e)
   {
       colorDialog1.ShowDialog();
       pictureBox1.BackColor = colorDialog1.Color;
   }
   int x, y, r;
  private void Form1_MouseDown(object sender, MouseEventArgs e)
   {
       x = e.X;
       y = e.Y;
       r = 5;
       timer1.Start();
   }
```

```
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    timer1.Stop();
}

private void timer1_Tick(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    SolidBrush cetka = new
         SolidBrush(pictureBox1.BackColor);
    g.FillEllipse(cetka, x - r, y - r, 2 * r, 2 * r);
    r += 3;
}
```



4. **Креирати апликацију која омогућава померање лоптице по форми после клика на једно од дугмади којима се бира смер померања.**



```
        int x, y, r = 20;

        private void Form1_Load(object sender, EventArgs e)
        {
            x = ClientRectangle.Width / 2;
            y = ClientRectangle.Height / 2;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            SolidBrush cetka = new SolidBrush(Color.Red);
            g.FillEllipse(cetka, x - r, y - r, 2 * r, 2 * r);
        }
//Pomeranje loptice se vrši klikom na dugme
        private void btLevo_Click(object sender, EventArgs e)
        {
            x -= 5;
            Refresh();
        }

        private void btDesno_Click(object sender, EventArgs e)
        {
            x += 5;
            Refresh();
        }

        private void btGore_Click(object sender, EventArgs e)
        {
            y -= 5;
            Refresh();
        }

        private void btDole_Click(object sender, EventArgs e)
        {
            y += 5;
            Refresh();
        }
```

5. Креирати апликацију која омогућава континуирано померање лоптице по форми у свим смеровима. Промена смера се врши кликом на одговарајуће дугме.

```
        int x, y, r = 20;
        int promenaX = 0, promenaY = 0;

        private void Form1_Load(object sender, EventArgs e)
        {
            x = ClientRectangle.Width / 2;
            y = ClientRectangle.Height / 2;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
```

```
        {
            Graphics g = e.Graphics;
            SolidBrush cetka = new SolidBrush(Color.Red);
            g.FillEllipse(cetka, x - r, y - r, 2 * r, 2 * r);
        }
//U ovoj aplikaciji klik na dugme samo menja smer kretanja...
        private void button1_Click(object sender, EventArgs e)
        {
            promenaX = -5;
            promenaY = 0;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            promenaX = 5;
            promenaY = 0;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            promenaX = 0;
            promenaY = -5;
        }

        private void button4_Click(object sender, EventArgs e)
        {
            promenaX = 0;
            promenaY = 5;
        }
//Pomeranje vršimo u pravilnim vremenskim intervalima
        private void timer1_Tick(object sender, EventArgs e)
        {
            x += promenaX;
            y += promenaY;
            Refresh();
        }
```