Osnove programskog jezika C#

Čas 4.

Delegati, događaji i interfejsi

Svojstva u C#

- Svojstvo daje (postavlja) informacije o objektu kome pripada. Svojstvo je jedan metod ili par metoda, međutim, pri upotrebi ono izgleda i ponaša se kao atribut.
- Na primer, kada pomerate neku kontrolu svojstvo ***Location*** vam daje informaciju o poziciji gde se ta kontrola nalazi. Tekstualni sadržaj je opisan svojstvom ***Text***, itd.

Vrste svojstava

- ***read-write*** *(dozvoljena promena i čitanje vrednosti)*
- ***read*** *(dozvoljeno je samo čitanje)*
- ***write*** *(dozvoljeno je samo upisivanje tj. promena vrednosti)*

Primer:

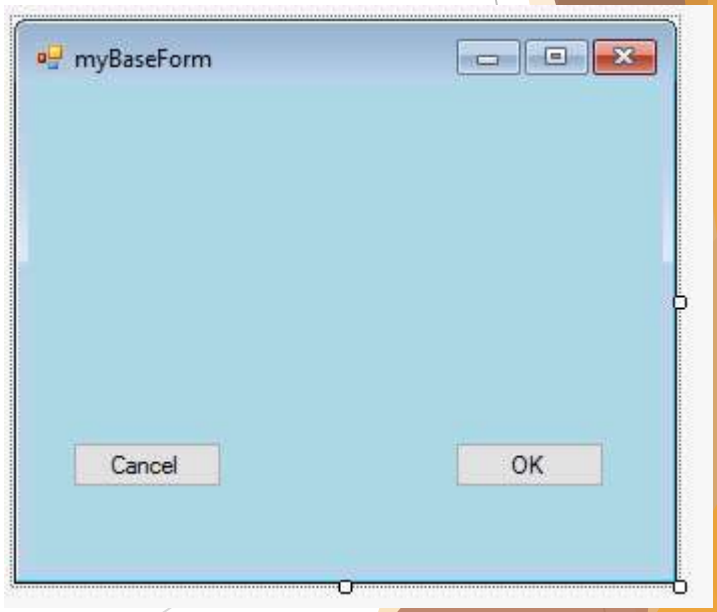
```
public class Property {
    public string Name{
        get{
            return name;
        }
        set{
            name = value;
        }
    }
    private string name;
}
```

```
class Class1{
    static void Main(){
        Console.Write("Name: ");
        Property property = new Property();
        property.Name = Console.ReadLine();      // set prop
        Console.WriteLine(property.Name);        // get prop
    }
}
```

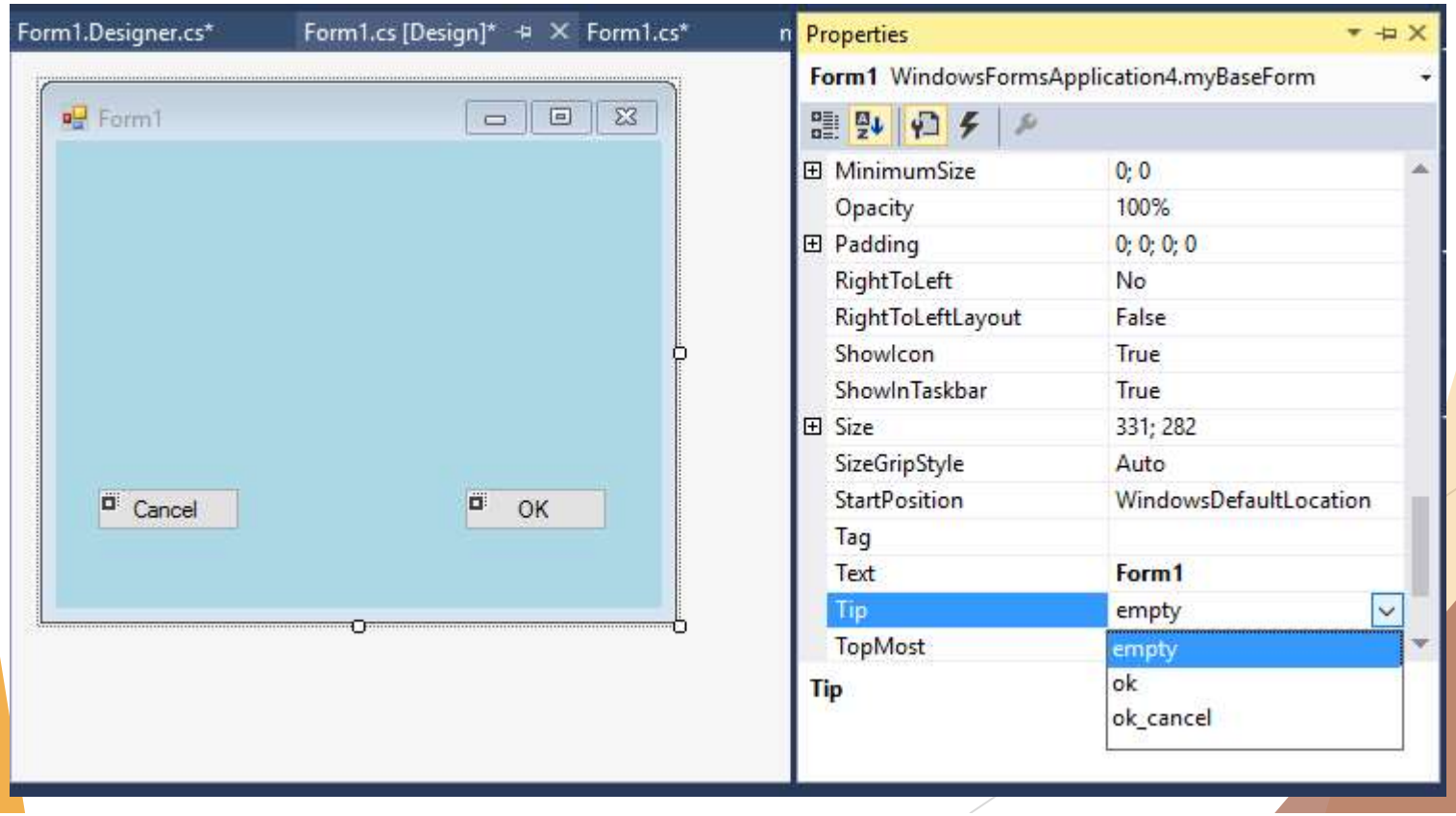
Primer

```
public partial class myBaseForm : Form {
    public myBaseForm(){
        InitializeComponent();
    }
    public enum tip { empty, ok, ok_cancel };

    tip tipForme;
    public tip Tip{
        get { return tipForme; }
        set {
            if (value == tip.empty){
                btnCancel.Visible = btnOK.Visible
                 = false;
            }
            else if (value == tip.ok){
                btnCancel.Visible = false;
                btnOK.Visible = true;
            }
            else if (value == tip.ok_cancel) {
                btnCancel.Visible = btnOK.Visible
                 = true;
            }
            tipForme = value;
        }
    }
}
```



```
public partial class Form1 :
myBaseForm
{
```



Apstraktne klase

- Ključna reč **abstract** omogućuje nam da stvorimo klase odnosno članove klase čije je namena da definišu svojstva koja moraju implementirati izvedene klase tj. ne-apstraktne klase. Apstraktne klase se ne mogu instancirati. Njihova svrha je da pruže zajedničku definiciju bazne klase koju nasleđuje više klasa. Apstraktne klase mogu definisati i apstraktne metode:

```
public abstract class A
{
    public abstract void DoWork(int i);
}
```

- Apstraktne metode nemaju implementaciju. Imaju samo tačku-zarez umesto normalne definicije u vitičastim zagradama.

```
public class B : A
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

- Izvedene klase apstraktne klase, ako i same nisu apstraktne, moraju implementirati sve apstraktne metode. Kad apstraktna klasa nasleđuje virtuelnu metodu iz bazne klase, ona može prespojiti virtuelnu metodu sa apstraktnom metodom:

```
public class D
{
    public virtual void DoWork(int i) {
        // Original implementation.
    }
}
public abstract class E : D
{
    public abstract override void DoWork(int i);
}
public class F : E
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

Interfejsi

- Interfejsi su apstraktne klase (čiste apstraktne klase) koje sadrže samo potpuno virtuelne metode. Njegova jedina svrha je **da deklariše skup metoda** a ne da ih realizuje.
- Ne sadrže polja! Elementi interfejsa ne mogu biti *static*.
- Ako sadrži svojstvo, klasa mora realizovati bar jednu od metoda svojstva *get* ili *set*. Može sadržati metode, svojstva, indeksere i događaje.
- Elementi interfejsa su implicitno *public abstract (virtual)*
- Klase i strukture mogu implementirati (naslediti) interfejse.
- Interfejsi mogu proširiti tj. naslediti druge interfejse.

Primer

```
public class myClass:IList
{

}
public interface IList : ICollection, IEnumerable
{
    int Add(object value);                  // metoda
    bool Contains(object value);
    bool isReadOnly { get; }                // svojstvo
    object this[int index] { get; set; }    // indekser
}
```

Nasleđivanje – proširenje sa novim metodama

Konverzija (kastovanje)

```
myClass c = new myClass();
IList l = (IList)c;
l.Add("asdf");

c = l as myClass;
c = (myClass)l;
```

DELEGATI

- Delegate = Tip, ali tip kojim se koristi metoda
- 1. Definicija tipa
- `delegate void Pozdrav(string sender);`
- 2. deklaracija promenljive
- `Pozdrav fPoz;`
- *Napomena: Zapazite sličnost sa deklaracijom objekta bilo koje klase.*
- Delegati su bezbedni pokazivači na funkcije...
  - Garantuju vrednost deklarisanog tipa. Prevodilac prijavljuje grešku ako pokušate da povežete funkciju koja ne odgovara deklarisanom tipu delegata.

Povezivanje metode sa objektom delegat

- Neka je metoda:
- ```
  void pozZdravo(string sender){
       Console.WriteLine("Zdravo " + sender);
  }
  ```

- Objekat se kreira kao i svi objekti referencnog tipa:
- ```
  fPoz = new Pozdrav(pozZdravo);
  ```

- A upotreba metode tj. pozivanje delgata tj. korišćenje objekta delegata:
- ```
  fPoz("Pera");
  ```
  ```

Rad sa različitim metodama

- Svaka metoda koja odgovara opisu koji delegat propisuje može biti pridružena promenljivoj.
- ```
  void pozDovidjenja(string sender){
       Console.WriteLine("Dovidjenja " + sender);
  }
  ```
- ```
  fPoz = new Pozdrav(pozDovidjenja);
  ```
- ```
  fPoz("Pera");
  ```

- Promenljiva koja je delegate može imati *null* vrednost, ako ni jedna metoda nije pridružena delegatu.
- Ako je *null* ne može biti pozivana. Promenljiva je klasnog tipa.
  ```

Rezime

- `new DelegateType(obj.Method)`

- Promenljiva koja je delegat čuva referencu na metodu. Promenljiva je zapravo objekat jedne klase koja se stvara i obezbeđuje rad sa metodama u stilu i na način kako se to radi sa objektima.

Osobine

- Metoda može biti **static**. U tom slučaju se ne koristi ime *obj.imemetode* već *imeklase.imemetode*
- Metoda **ne može biti *abstract***. Može biti *virtual, override or new. Napomena: apstraktna metoda ne poseduje telo već samo opis metode.*
- *Potpis metode mora odgovarati DelegateType*

Višeznačni delegati

- Promenljiva koja je delegat može čuvati tj. sadržati više vrednosti u isto vreme.
- ```
  Pozdrav fPoz;
  fPoz = new Pozdrav(pozZdravo);
  fPoz += new Pozdrav(pozDovidjenja);
  ```
- ```
  fPoz("Pera"); // “Zdravo Pera”, ”Dovidjenja Pera”
  ```
- Zapazite da se poslednja metoda poslednja poziva.

Moguća kontrola višeznačnosti

- ```
  fPoz -= new Pozdrav(Zdravo);
  ```
- ```
  fPoz("Pera"); // ”Dovidjenja Pera”
  ```

Event – promenljiva tipa delegata

- Objekat koristi događaj da bi obavestio drugi objekat da se nešto desilo.
- Događaji su objekti tipa delegata:

- `public event nazivDelegata nazivDogađaja`

Tip događaja

```
class Model {
     public event Notifier notifyViews;
     public void Change() {
         //...
         notifyViews("Model");
     }
 }
 class View1 {
     public View1(Model m) { m.notifyViews += M_notifyViews; }

     private void M_notifyViews(string sender)
     {
         Console.WriteLine(sender + " was changed - view1");
     }
 }
 class View2
 {
     public View2(Model m) { m.notifyViews += M_notifyViews2; ; }

     private void M_notifyViews2(string sender)
     {
         Console.WriteLine(sender + " was changed - view2");
     }
 }
```

```
class Program
   {
        static void Main(string[] args)
        {
            Model m = new Model();
            new View1(m);
            new View2(m);
            m.Change();
        }
    }
```

Primer i zadaci

- Prva forma ima dugme „promeni boju“ i dugme “kontrolni ekran”.
  - Pritiskom na dugme “promeni boju” menja se boja forme.
  - Pritskom na dugme „ kontrolni ekran“ otvara se druga forma sa tri labele u kojima će biti prikazivane komponente boje prve forme.
- Koristeći delegate i događaje omogućiti da labele sadrže RGB komponente boje koja je na prvojformi.

PrimerSopsDogadjaji1

Zadatak za vežbanje:

- Napisati aplikaciju koja može da otvori više identičnih formi sa jedne roditeljske. Sve novootvorene forme imaju jedno polje za unos teksta (textbox).
- Roditeljska forma ima jedno dugme za otvaranje forme i onoliko labela koliko ima otvorenih formi iz nje. U svakoj labeli treba da bude naziv novootvorene forme i tekst u textbox-u te forme.
- Problem rešiti koristeći događaje.

Anonimni delegati

```
public bool testiranje(string x)
{
    return x == "test";
}
```

```
List<string> a = new List<string>();
a.Add("pera"); a.Add("test"); a.Add("mika");
a.Add("test"); a.Add("test");

List<string> rez0 = a.FindAll(testiranje);
List<string> rez1 = a.FindAll(delegate (string x) { return x == "test"; });

//INFORAMTIVNO
List<string> rez2a = a.FindAll(new Predicate<string>(testiranje));
List<string> rez2b = a.FindAll(x => x == "test");
```

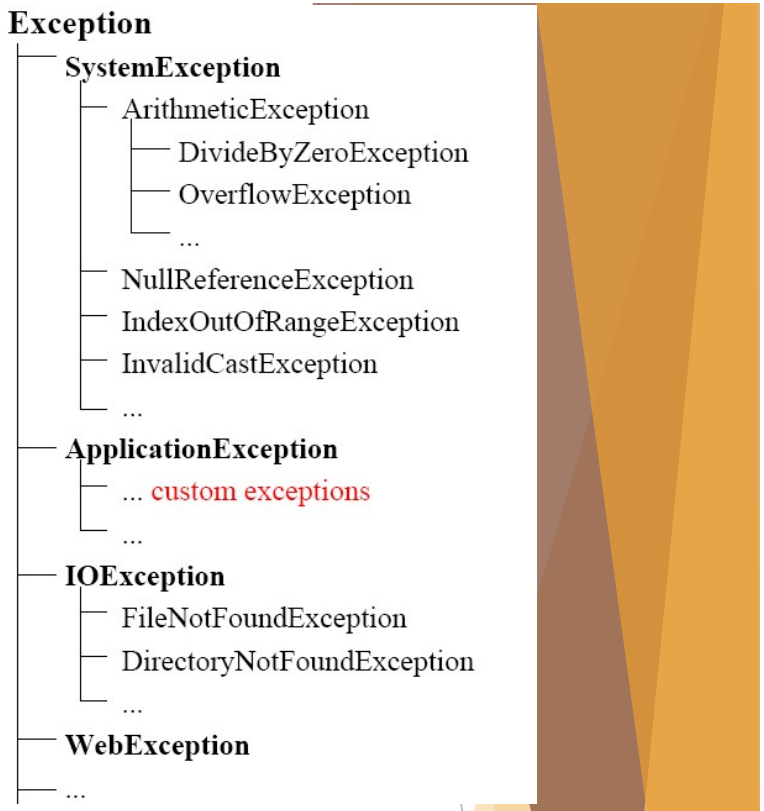
Izuzeci

- Izuzeci mogu biti različitog porekla. Sam programer može da prijavi određenu grešku kao izuzetak, ali i različite programske biblioteke, u zavisnosti od toga koju koristimo, prijavljuju određene probleme kao izuzetke.
- Kada se izuzetak desi, on se prijavljuje programskom bloku višeg nivoa, tj. roditeljskom bloku, u formi nekog podatka koji ga opisuje. Izuzetak se može prijaviti u obliku teksta „Desio se izuzetak“, u obliku celog broja, objekta neke korisničke klase itd. dokle god taj podatak na dobar način opisuje spomenuti izuzetak. Obično se kaže da je izuzetak „izbačen“, kada se govori o prijavljivanju izuzetka roditeljskom bloku.

- Roditeljski blok može da odabere da ignoriše bilo kakve izuzetke, i u tom slučaju izuzetak će automatski biti prosleđen **njegovom** roditeljskom bloku. Ukoliko se, međutim, roditeljski blok unapred pripremi za izuzetke određenog tipa, kaže se da on „hvata“ izuzetak i od njega zavisi kako će se izuzetkom rukovati.
- Ukoliko nijedan blok ne uhvati izuzetak, on se penje do prvog, početkog programskog bloka. Ako ga ni on ne uhvati, operativni sistem obično reaguje nasilnim prekidanjem rada programa; to se obično izvodi slanjem signala ABORT. Signal ABORT se može ignorisati, ali čak i tada proces ostaje u stanju poremećene funkcionalnosti pa se to ne preporučuje. Iz ovih razloga se preporučuje pažljiva obrada izuzetaka, da bi se problem mogao prijaviti korisniku i, eventualno, situacija popraviti.

try, catch, finally

```
FileStream s = null;
try {
    s = new FileStream(curName, FileMode.Open);
    ...
} catch (FileNotFoundException e) {
    Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
    Console.WriteLine("some IO exception occurred");
} catch {
    Console.WriteLine("some unknown error occurred");
} finally {
    if (s != null) s.Close();
}
```



- `catch` – provere izuzetaka u sekvencijalnom redosledu
- Parametar `Exception` može biti izostavljen, onda je podrazumevani `System.Exception`
- Izuzeci moraju biti izvedeni iz klase `System.Exception`

System.Exception

Svojstva

- e.Message - poruka u kao tip string
- e.StackTrace - trag metoda
- e.Source - objekat koji izbacuje izuzetak
- e.TargetSite - metod objekta koji izbacuje izuzetak
- ...

Metode

- e.ToString() vraća naziv izuzetka

BLOK *finally*

- Nije obavezan
- Blok koda koji će uvek biti izvršen bez obzira na to da li u ispitnom bloku nastaje izuzetak ili ne. ***finally*** blok će se uvek izvršiti.

Izbacivanje izuzetaka

- **U slučaju neispravne operacije (implicitni izuzeci)**
  - Deljenje sa 0
  - “Index overflow”
  - “Acess via a null reference”
  - …
- **U slučaju namernog izbacivanja izuzetka (explicitno)**
  - `throw new FunnyException(10);`
  - ```
    class FunnyException:ApplicationException{
       public int errorCode;
       public FunnyException(int x){ errorCode = x;}
    }
    ```
    ```

Random

- Predstavlja generator pseudo-slučajnih brojeva.
- Konstruktori:
  - `Random(int)` – inicijalna vrednost za generisanje sluč.brojeva
  - `Random()` – inicijalna vrednost je vremenski *tick*
- `Random.Next()` – vraća nenegativni slučajan broj.
- `Random.Next(int)` – vraća nenegativan broj manji od definisanog maksimuma.
- `Random.Next(int, int)` – vraća slučajan broj u nekom opsegu.
- `public virtual double NextDouble()`
  - Vraća broj dvostruke preciznosti u pokretnom zarezu koji je veći od 0.0 a manji od 1.0.

PRIMER

RandomObjectDemo.cs

Primer RandomObjectDemo.cs

- Rezultati testa

- FixedSeedRandoms(123);
- FixedSeedRandoms(123);

- FixedSeedRandoms(456);
- FixedSeedRandoms(456);

- Random numbers from a Random object with seed = 123:
- 2114319875 1949518561 1596751841 1742987178 1586516133 103755708
- 0.01700087 0.14935942 0.19470390 0.63008947 0.90976122 0.49519146

- Random numbers from a Random object with seed = 123:
- 2114319875 1949518561 1596751841 1742987178 1586516133 103755708
- 0.01700087 0.14935942 0.19470390 0.63008947 0.90976122 0.49519146

- Random numbers from a Random object with seed = 456:
- 2044805024 1323311594 1087799997 1907260840 179380355 120870348
- 0.21988117 0.21026556 0.39236514 0.42420498 0.24102703 0.47310170

- Random numbers from a Random object with seed = 456:
- 2044805024 1323311594 1087799997 1907260840 179380355 120870348
- 0.21988117 0.21026556 0.39236514 0.42420498 0.24102703 0.47310170

Zadatak za samostalan rad

- Koristeći random generator osmisliti i realizovati zaštitni koder i dekoder za snimanje/prenos podataka.