Pokročilé objektové technológie

2

Základy jazyka C# 2/2

KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Indexery

- Syntaktické rozšírenie
- Umožňuje pracovať s objektom ako keby bol pole
- V jazyku Java sa miesto indexeru používali metódy get a put/set
- Operátor [] v C++
- Definícia podobne ako pre vlastnosti
- Názov je vždy „this“

Indexery v jazyku Java

```
public class Vektor {
   private double x, y;
   public double get(int i) {
      switch (i) {
         case 0:
            return x;
         case 1:
            return y;
         default:
            return 0;
      }
   }
   public void set(int i, double v) {
      switch (i) {
         case 0:
            x = v; break;
         case 1:
            y = v; break;
      }
   }
}
```

Indexery

```
class Vektor {
   private double x, y;
   public double this[int i] {
      get {
         switch (i) {
            case 0:
               return x;
            case 1:
               return y;
            default:
               return 0;
         }
      }
      set {
         switch (i) {
            case 0:
               x = value; break;
            case 1:
               y = value; break;
         }
      }
   }
}
```

Indexery

```
...

Vektor v1 = new Vektor();
Vektor v2 = new Vektor();
v1[0] = 10;
v1[1] = 5;
v2[0] = v1[0]*4;
...
```

Indexery

- Index akýkoľvek dátový typ
- Návratový typ takisto akýkoľvek
- Možnosť vynechať get alebo set časť
- Aj ako viacrozmerné polia
- Viacej indexerov v jednej triede
- Nemôže byť použitý vo foreach

Preťažovanie metód

- Overloading
- Viac definícii jednej metódy
- Rovnako ako v jazyku Java
- Možnosť obmedziť používanie pomocou nepovinných parametrov

Nepovinné parametre metód

- Parametrom je priradená defaultná hodnota
- Musia byť za povinnými parametrami

```
public static int Calc(int a, int b = 10, int c = 2)
{
    return (a + b) * c;
}

Calc(55);
Calc(55, 10);
Calc(55, 8, 3);
```

Pomenované argumenty metód

- Argumenty je možné predávať v ľubovoľnom poradí cez názvy parametrov

```
public static int Calc(int a, int b = 10, int c = 2)
{
    return (a + b) * c;
}

Calc(55, c: 3);
Calc(c: 3, a: 1, b: 2);
```

Dedičnosť

- Jednoduchá
- Preddefinovaný predok
- base miesto super

```
public class DopravnyProstriedok : object {...} // alebo
                                                 // System.Object

public class Auto : DopravnyProstriedok {...}
```

System.Object

- Spoločný predok všetkých objektov
- Základné metódy:
  - string ToString()
  - int GetHashCode()
  - bool Equals(object obj)
  - bool ReferenceEquals(object objA, object objB)
  - Type GetType()
  - object MemberwiseClone()
  - void Finalize()

Prekrývanie metód

- Overriding
- virtual
- override
- Nemôžu byť statické metódy a členské premenné
- Zhodná signatúra

```
DopravnyProstriedok
public virtual bool JeOsobne()
{...}

Auto
public override bool JeOsobne()
{...}
```

Abstract

- Nie je možné vytvoriť inštanciu
- Musí byť prekrytá
- Členy
  - Triedy
  - Metódy
  - Vlastnosti

Abstract

```
public abstract class DopravnyProstriedok {

   public abstract bool JeOsobne(); // uz nepotrebuje telo
   public abstract int MaxRychlost {
      get;
      set;
   }
}

public class Auto : DopravnyProstriedok {

   public override bool JeOsobne() {...}
   public override int MaxRychlost {
      get { ... }
      set { ... }
   }
}
```

Partial

- Kľúčové slovo **partial**
- Rozdelenie definície
- Spájané pri preklade
- Obmedzenia:
  - Všade partial
  - Iba v rámci zostavenia
  - Zhodné názvy

- Používané napr. pri GUI (generovaná časť)

Rozhranie

- Interface
- Môže obsahovať deklarácie:
  - Metód
  - Vlastností
  - Indexerov
  - Udalostí
- Členy nemajú modifikátory
  - Vždy verejné
  - Nie deklarácia ako virtuálne alebo statické

Implementácia rozhraní

- Podobne ako pri dedičnosti
- Implementovať všetky členy
- Trieda aj rozhranie môžu implementovať súčasne viacej rozhraní

Rozhranie

```
public interface IDopravnyProstriedok {

   bool JeOsobne();  // uz nepotrebuje telo
   int MaxRychlost {
      get;
      set;
   }
}

public class Auto : IDopravnyProstriedok {

   public bool JeOSobne() { return false; }
   public virtual int MaxRychlost {
      get { ... }
      set { ... }
   }
}
```

Delegáti

- Delegates
- Nový typ objektu – potomok System.Delegate
- Na predávanie metód
  - Iným metódam
  - Triedam
  - Objektom
- Zapuzdrenie metódy do nového typu objektu
- Obsahujú podrobnosti o metóde
- Anonymné volanie

Delegáti - deklarácia

- Aký typ metódy bude obsahovať
- Presná signatúra
- Definícia novej triedy

```
public delegate void PrisielZakaznikEvent(Zakaznik zakaznik);
```

Vytvorenie inštancie delegáta

- Konštruktor – jeden argument
  - Metóda
  - Iný delegát
- Nemôže byť referencia na:
  - property, indexer, užívateľsky definovaný operátor, inštančný konštruktor, deštruktor, static konštruktor

Použitie delegáta

```
public void Zobraz(List<Osoba> zoznam)
{
    zoznam.Sort(new Comparison<Osoba>(this.Porovnaj));
    foreach (Osoba o in zoznam)
    {
        Console.WriteLine(o);
    }
}

public int Porovnaj(Osoba prva, Osoba druha)
{
    ...
}
```

Použitie delegáta

```
public void Zobraz(List<Osoba> zoznam)
{
    // boxing
    zoznam.Sort(this.Porovnaj);
    foreach (Osoba o in zoznam)
    {
        Console.WriteLine(o);
    }
}

public int Porovnaj(Osoba prva, Osoba druha)
{
    ...
}
```

Aktivácia delegáta

- Nutná jeho inštancia
- Spustí priradenú metódu
  - Predá jej parametre
  - Vráti výslednú hodnotu
  - Môže byť ukončený aj výnimkou

Aktivácia delegáta

```
delegate double DoubleFunc(double x);

class A
{
   DoubleFunc f = new DoubleFunc(Square);

   public static double Square(double x) {
      return x * x;
   }
}
...
double ret = f(4.0);
...
```

Anonymné metódy

```
button1.Click += new System.EventHandler(button1_Click);

private void button1_Click(object sender, EventArgs e)
{
   MessageBox.Show("Ahoj");
}
```

```
button1.Click += delegate(object sender, EventArgs e)
                 {
            MessageBox.Show("Ahoj");
                 };
```

```
private int[] integers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int[] evenIntegers = Array.FindAll(integers,
                                   delegate(int integer)
      {
         return (integer % 2 == 0);
      } );
```

Lambda výrazy

```
List<int> listOfInts = new List<int>() { 1, 2, 3, 4, 5 };

//C# 2.0
int result = listOfInts.Find(delegate(int i)
{
   return i > 2;
});

//C# 3.0
int result = listOfInts.Find(i => i > 2);
```

Viacnásobní delegáti

- Multicast delegate
- Volanie viacerých metód
- Môže zapúzdrovať viacej metód
- Postupné volanie
- Malo by vracať void
- Operátory +, +=, -, -=
- Nedefinované poradie

Udalosti

- Events
- Špeciálna forma delegátov
- Metóda udalosti – Event handler
  - Na strane spotrebiteľa
  - Nesmie vracať žiadnu hodnotu
  - Predpísané vstupné argumenty

```
void OnClick(object sender, EventArgs e) // argument e je mozne
{             // aj odvodit
   // kod na obsluhu udalosti
}
```

Udalosti

- Strana zasielateľa
  - Špeciálna podoba viacnásobného delegáta
  - Verejný člen triedy - event

```
public class ZakaznikEv {
...
  public delegate void PrisielZakaznikEventHandler(object
      sender,  EventArgs e);

  public static event PrisielZakaznikEventHandler
   PrisielZakaznik;
...
```

Spotrebiteľ

```
public class BankaEv {
   private ZakaznikEv zakaznik;

   public BankaEv() {
      ZakaznikEv.PrisielZakaznik += PrichodZakaznika;
   }

   private void PrichodZakaznika(object sender,
   EventArgs e)
   {
      ZakaznikEv zakaznik = sender as ZakaznikEv;
      ...
   }
...
```

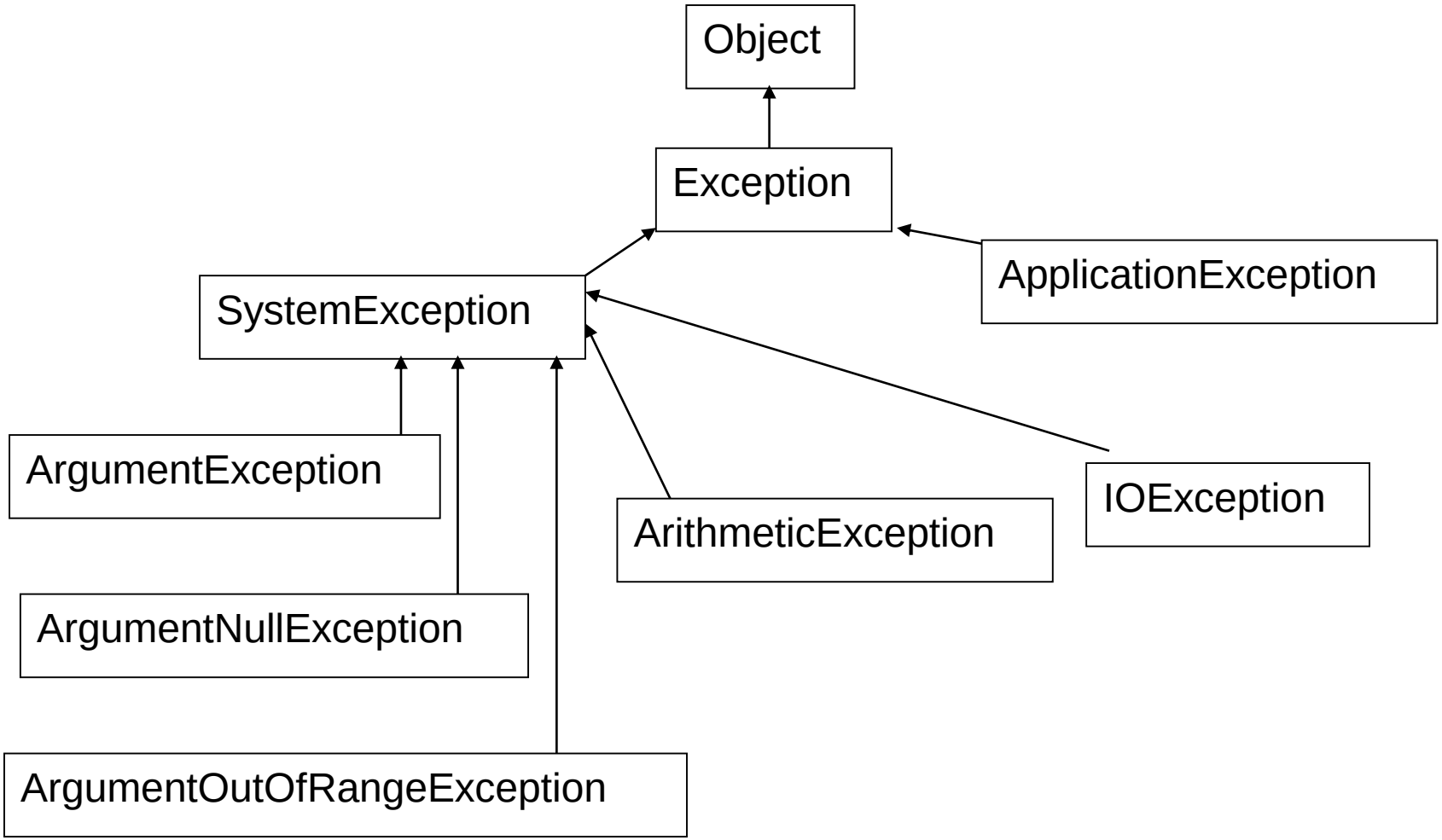
Vyvolanie

```
public class ZakaznikEv {
...
  public delegate void PrisielZakaznikEventHandler(object
      sender,  EventArgs e);

  public static event PrisielZakaznikEventHandler
   PrisielZakaznik;
...

  protected virtual void OnPrisielZakaznik(EventArgs e) {
     if (PrisielZakaznik != null)
        PrisielZakaznik(this, e);
  }
```

Výnimky

- Na ošetrenie chybových stavov
  - Nenašiel sa súbor
  - Spadla sieť
  - …
- Rovnako ako v Java
- Preddefinované výnimky
- Vlastné výnimky
- Netreba throws v hlavičke, obdoba nekontrolovaných výnimiek

Bázové triedy výnimiek



Generická trieda

```
static void Main()
{
   List<Person> people = new List<Person>();
   people.Add(new Person());

   List<Car> cars = new List<Car>();
   cars.Add(new Person());    // !!! Chyba v dobe prekladu

   List<int> ints = new List<int>();
   ints.Add(5);           // ziadna operacia boxingu
   int I = ints[0];       // ziadna operacia unboxingu
}
```

System.Collections.Generic

Kolekcia	Java	Význam
Dictionary<K, V>	HashMap<K, V>	Kolekcia dvojíc kľúč-hodnota
List<T>	ArrayList<E>	Dynamické pole
Queue<T>	Queue<E>	FIFO
Stack<T>	Stack<E>	LIFO
LinkedList<T>	LinkedList<E>	Obojsmerný zreťazený zoznam
Collection<T>	AbstractCollection<E>	Základná trieda kolekcií

List<T>

- Príklad použitia z BCL:

```
public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>,
IList, ICollection, IEnumerable
{
   public List();
   public List(IEnumerable<T> collection);
   public List(int capacity);
   public int Capacity { get; set; }
   public int Count { get; }
   public T this[int index] { get; set; }
   public void AddRange(IEnumerable<T> collection);
   public ReadOnlyCollection<T> AsReadOnly();
   public int BinarySearch(T item);
   ...
}
```

Generické triedy a štruktúry

- Podobne ako v Java
- Postup vytvárania zhodný
- Typový parameter za názvom triedy
- Možné používať vnútri triedy:
  - Dátové členy
  - Parametre funkčných členov
  - Parametre vnorených členov
- Nepodporované operátory

Generické rozhrania

```
public interface IBinaryOperations<T> {
    T Add(T args1, T arg2);
    T Substract(T arg1, T arg2);
    ...
}

public class BasicMath : IBinaryOperations<int> {
    public int Add(int arg1, int arg2)
    {
        return arg1 + arg2;
    }
}
```

Generické metódy

- Typový parameter **za názvom** metódy
- Pri volaní:
  - S uvedením typového parametru
  - Bez uvedenia

Generické metódy

```
static void swap<T>(ref T a, ref T b) {
   T temp;
   temp = a;
   a = b;
   b = temp;
}

int a = 10, b = 90;
Swap<int>(ref a, ref b);   // alebo Swap(ref a, ref b);

void DisplayType<T>() {
   Console.WriteLine("Zaslanou triedou je {0}", typeof(T));
}

DisplayType<int>();
DisplayType(); // !!! chyba
```

Generické metódy

```
public strut Point<T> {
   private T xPos, yPos;
   public Point(T xVal, T yVal) {
      xPos = xVal; yPos = yVal;
   }
   public T X {
      get { return xPos;}
      set { xPos = value;}
   }
   public  void ResetPoint() {
      xPos = default(T);
      yPOs = default(T);
   }
}

Point<int> p = new Point<int>(10, 10);
Point<double> d = new Point<double>(1.4, 2.5);
```

Klauzula where

Generické obmedzenie	Význam
where T : struct	Typový parameter musí byť hodnotový typ
where T : class	Typový parameter musí byť referenčný typ
where T : new()	Typový parameter musí mať štandardný konštruktor
where T : NazovTriedy	Typový parameter musí byť odvodený od NazovTriedy
where T : NazovRozhrania	Typový parameter musí implementovať rozhranie NazovRozhrania
where T : U	Typový parameter T musí byť typu U

Klauzula where

```
class MojaTrieda1<T> where T : new() {…}

class MojaTrieda2<T> where T : class, IDaSaNakreslit, new() {…}

class MojaTrieda3<T> : MojaZakladna where T : struct {…}

class MojaTrieda4<K, T> where K : new() where T : IComparable<T>

// chyba
class MojPotomo<T> : MojaTrieda1<T> {…}
```

Rušenie objektov

- Spravované zdroje – Garbage Collector
- Nespravované zdroje – súbory, sieťové spojenia
- Object.Finalize()
- Dopad na výkon

Rušenie objektov

- Deštruktor - volaný pri rušení inštancie
- Ukončenie platnosti objektu a vymazanie z pamäti automatickou správou – časový posun
- Dvojstupňová implementácia:
  - Implementácia rozhrania IDisposable (Dispose)
  - Definícia deštruktora

IDisposable

- Dispose()
  - Uvoľniť všetky zdroje čo vlastní
  - Aj zdroje predkov
  - Umožniť viacnásobné volanie

Implementácia

```
class MojaTrieda : IDisposable
{
    // nemala by byť virtualna
    // potomkovia by nemali mať možnosť prekryť túto metódu
    public void Dispose() {
        ...
        // uvolnenie zdrojov
    }
}
```

Použitie

```
using (MojaTrieda m = new MojaTrieda())
{
    ...
    // vykonaj co treba
}
// automaticky sa zrusi
```

```
MojaTrieda m = new MojaTrieda()

try
{
    ...
    // vykonaj co treba
}
finally
{
    m.Dispose();
}
```

Nullable typy

- hodnotové typy použité ako referenčné
- Nullable<hodnotovy_typ>

```
int? cislo = 5; // Nullable<int>
cislo = null;
```

Operátor ?? (null-coalescing operator)

```
int? x = null;
int y = x ?? -1;    // y = x != null ? x : -1
```

```
int i = GetNullableInt() ?? default(int);
```

```
string s = GetStringValue();
Console.WriteLine(s ?? "Reťazec s je rovný null");
```

Extension metódy

- „Pridávanie metód“ do existujúcich tried
- Definované ako statické, volané ako inštančné

```
public static class MyExtensions {
   public static int WordCount(this String str) {
      return str.Split(new char[] { ' ', '.', '?' },
        StringSplitOptions.RemoveEmptyEntries).Length;
   }
}

string s = "Hello Extension Methods";
int i = s.WordCount();
```