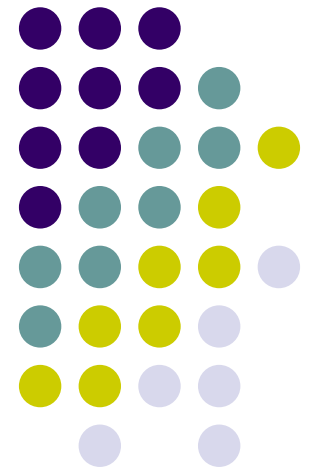


Komponentní technologie

Ing. Marek Běhálek
Katedra informatiky FEI VŠB-TUO
A-1018 / 597 324 251

<http://www.cs.vsb.cz/behalek>
marek.behalek@vsb.cz





Obsah kapitoly

- Komponentní technologie
- Java Beans
- Technologie COM a tvorba komponent v C/C++
- Úvod do architektury .NET
- Stručný tutoriál jazyka C#
- Komponenty v .NET implementované v C#

Komponenty - Motivace



- Vývoj
 - Opakovaná použitelnost
 - Snadnost testování
 - Možnost specializace výrobců
- Distribuce
 - Rychle uvedení na trh
 - Nezávislost na dodavateli
- Údržba
 - Snížení nákladů na údržbu
 - Zaměnitelnost – tlak odběratelů na standardizaci

Komponenty - Inspirace z jiných oborů



- Stavebnictví
 - Výstavba z modulů, panely
- Automobilový průmysl
 - Specializace ve výrobě, zaměnitelnost
- Elektronika
 - Základní součástky, moduly, funkční bloky



Komponenty - Aplikace v IT

- Technické vybavení
 - Paměť, procesory, základní desky
 - Periferní zařízení – PnP, ovladače
 - Komunikační prvky
- Programové vybavení
 - Grafické uživatelské rozhraní – Swing
 - Distribuované aplikace – CORBA, EJB, .NET, ...
 - Databáze
 - Informační systémy

Komponenty - Vývoj přístupů k tvorbě programů (1)



- Monolitické programy
 - upřednostnění úspornosti kódu na úkor srozumitelnosti a přehlednosti
- Strukturované metody
 - hierarchie, abstrakce
 - dynamické knihovny (DLL)
 - vzdálené volání procedur (RPC)
 - modulární programování (Modula-2)

Komponenty - Vývoj přístupů k tvorbě programů (2)



- Objektově orientované technologie
 - zapouzdření – „černá skříňka“
 - dědění
 - polymorfismus
 - znovupoužitelnost
- Komponentní technologie
 - rozhraní
 - standardy

Komponenty - Co je to komponenta? (1)



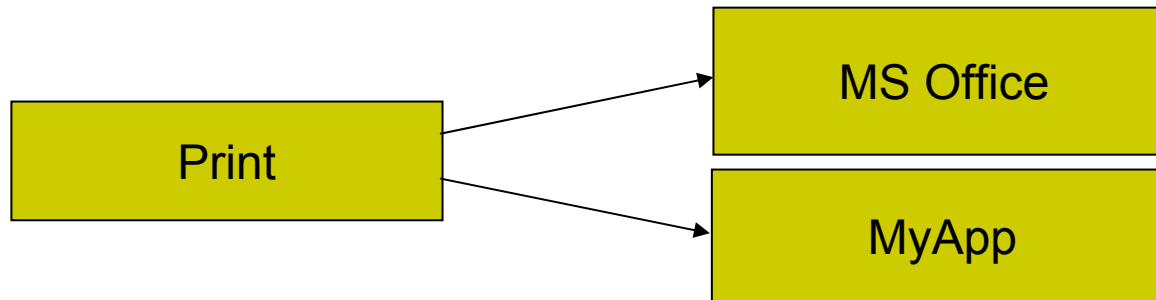
- Stavební jednotka se smluvně definovanými:
 - rozhraními;
 - explicitními kontextovými vazbami



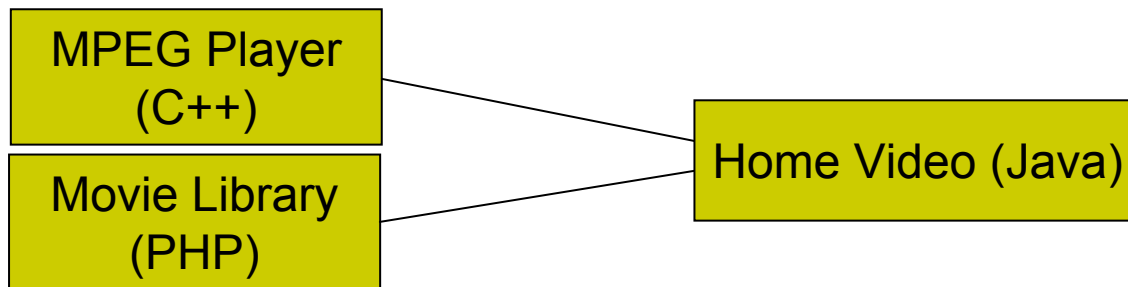
Komponenty - Co je to komponenta? (2)



- Může být použita nezávisle na
 - prostředí, pro které byla vytvořena,



- prostředí, ve kterém byla vytvořena.



Komponenty - Co je to komponenta? (3)



- Je určena pro integraci třetí stranou
 - Autor komponenty
 - Neví, kdo a k čemu bude jeho komponenty využívat
 - Musí dodržet stanovené rozhraní
 - Autor aplikace
 - Neví, kdo bude dodávat komponenty
 - Komunikuje přes stanovené rozhraní
 - **Integrátor**
 - Propojí aplikaci s vhodnými komponentami

Komponenty - Požadavky na komponenty



- Úplná dokumentace
- Důkladné testování
- Robustní kontrola platnosti vstupů
- Vracení dostatečných informativních chybových zpráv
- Vycházet s toho, že komponenta bude použita k předem nepředpokládaným účelům.

Komponenty - Specifikace komponenty



- Stav
 - Vlastnosti – čtení, nastavení
- Chování
 - Operace – volání, parametry, výsledek
- Interakce s okolím
 - Události – registrace, oznámení

Komponenty - Životní cyklus komponenty (1)



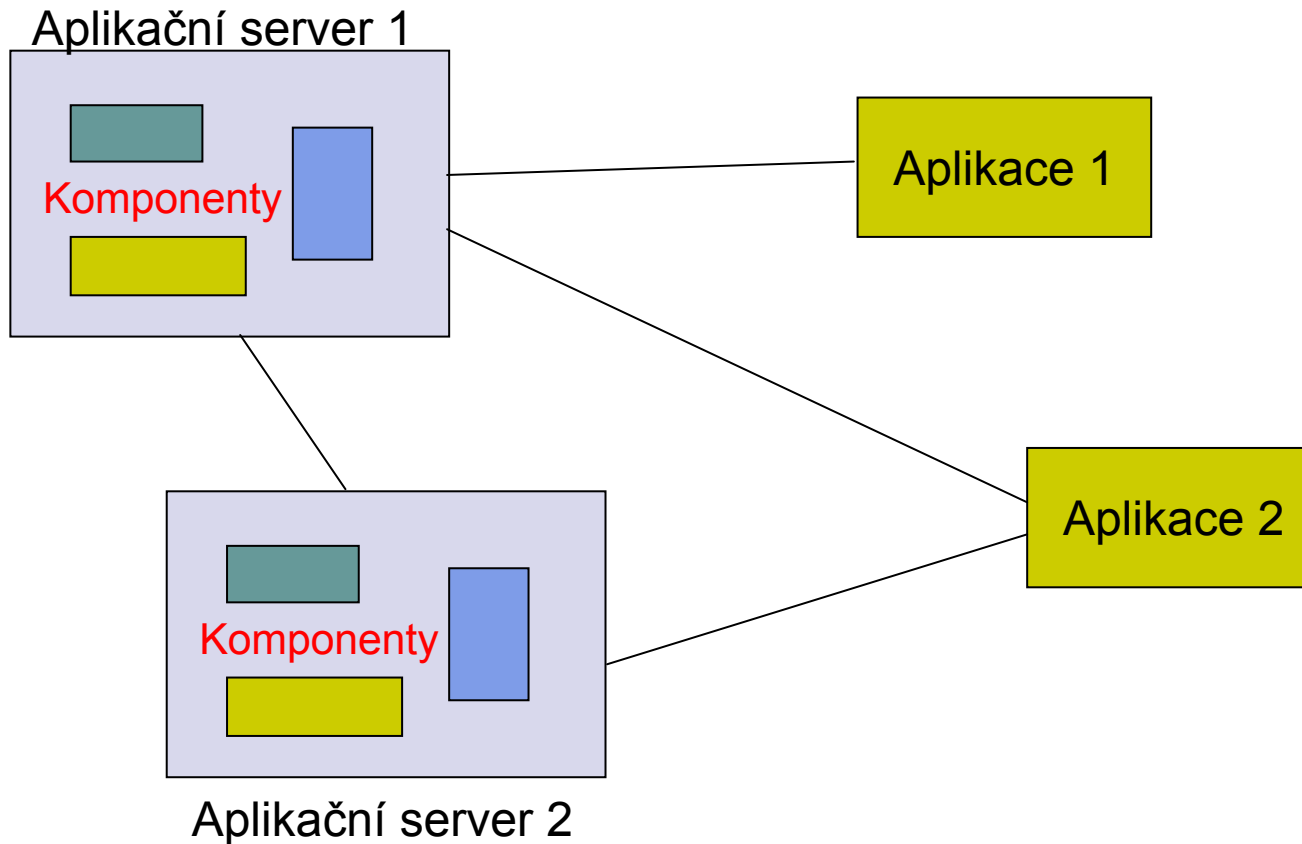
- Tvorba komponenty
 - Standardy – CORBA, COM+, EJB, .NET
 - Binární kompatibilita – nezávislost na jazyce
- Publikace rozhraní
 - Dokumentace – pro člověka
 - Introspekce – součást komponenty, klientská aplikace může číst metadata

Komponenty - Životní cyklus komponenty (2)



- Šíření komponenty
 - registrace komponenty, adresářové služby
 - LDAP, JNDI, UDDI
- Vyhledávání komponenty
 - identifikace komponenty, pozdní vazba
- Tvorba aplikace
 - Podpora IDE – přístup jako k interním objektům

Komponenty - Architektura komponentně orientovaných systémů

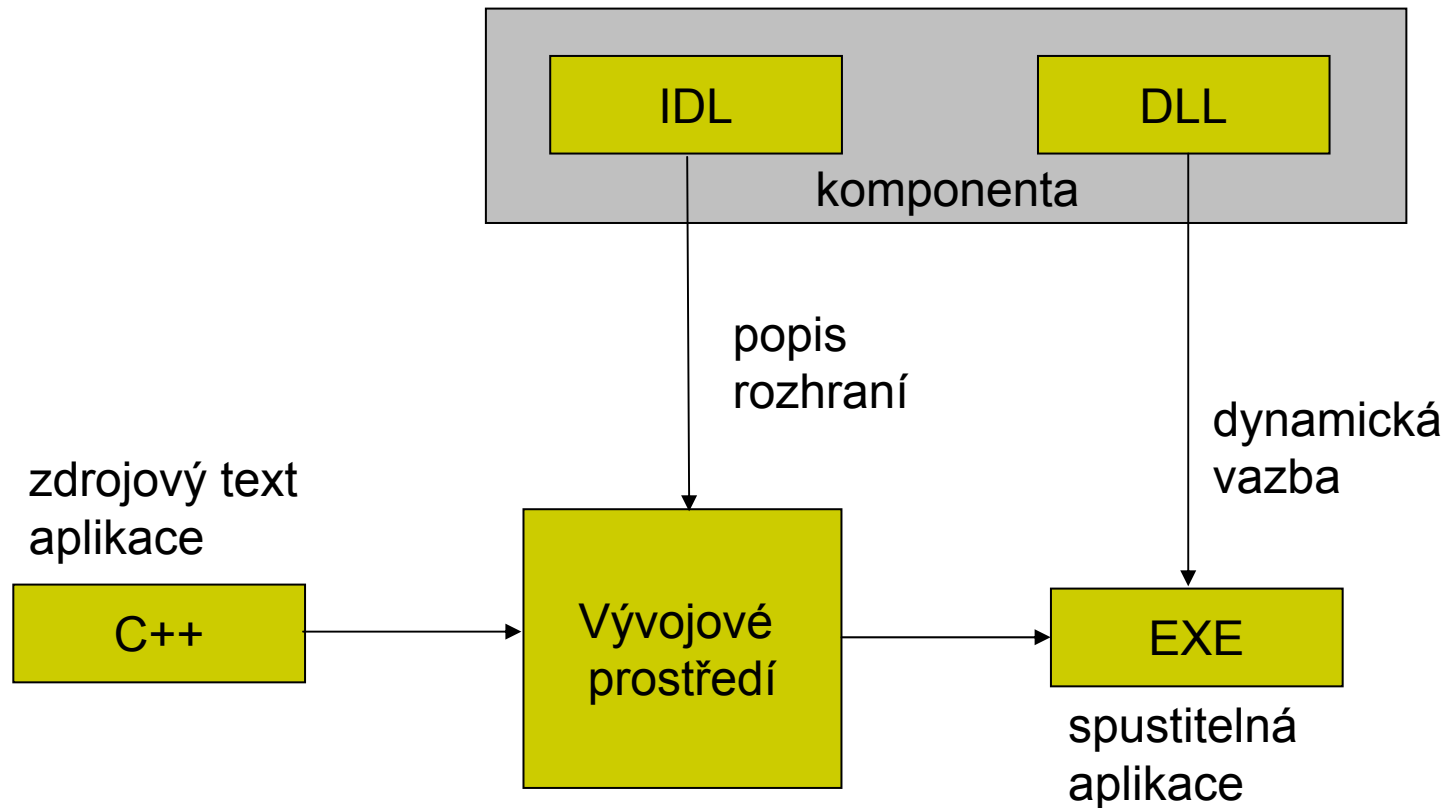


Komponenty - Aplikační server



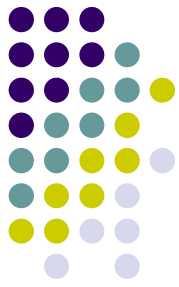
- Prostředí pro běh softwarových komponent
 - Distribuované prostředí
 - Sdílení dat
 - Synchronizace
 - Zabezpečení
- Příklady
 - JBoss, Jakarta Tomcat, BEA Weblogic, Citrix, Meta Frame, IBM WebSphere, Oracle AS, Java Systems Sun AS, ...

Komponenty - Příklad tvorby aplikací



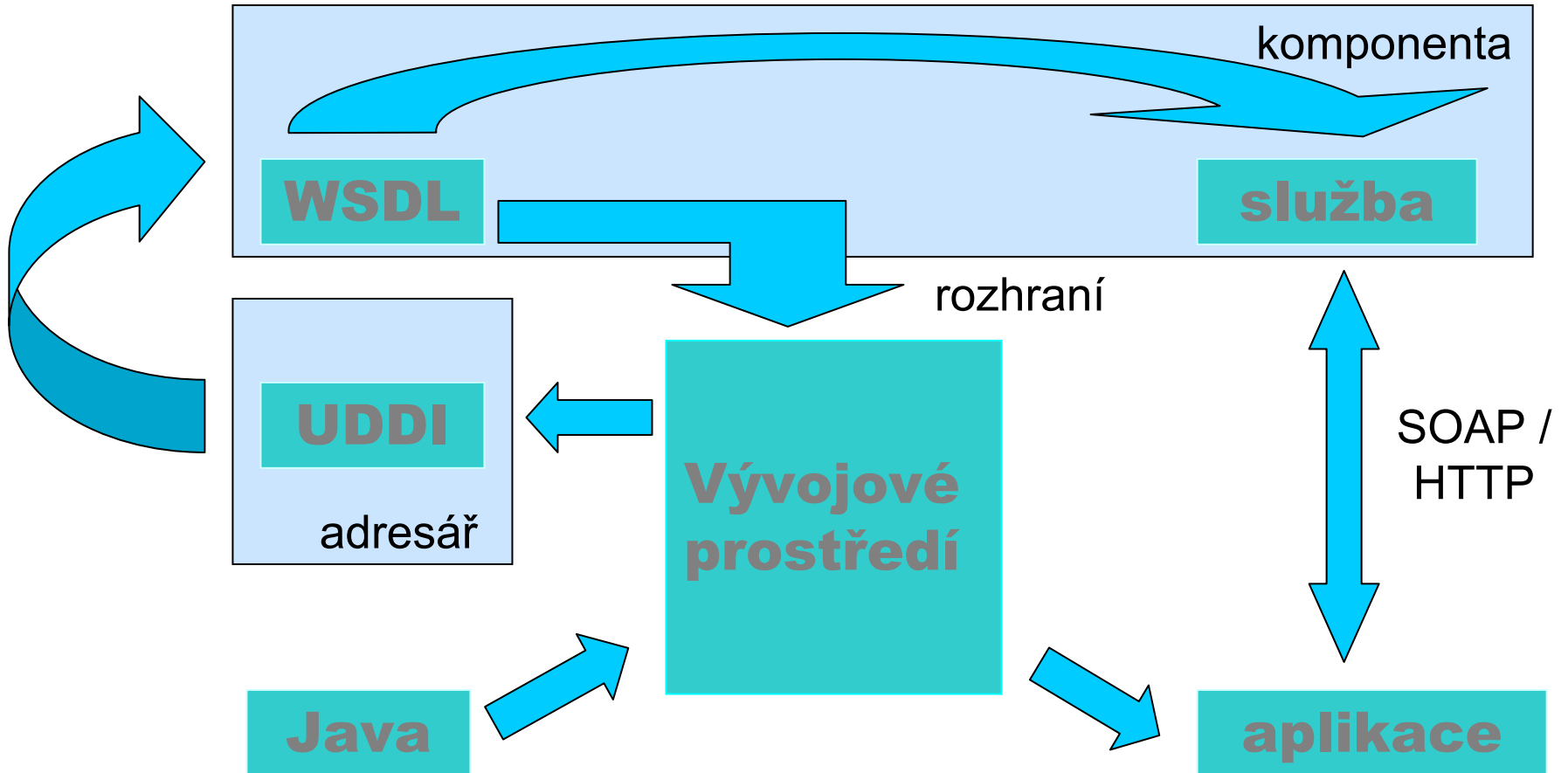
Komponenty – Webové služby

(1)



- **Webové služby** (Web Services)
 - komponenty přístupné přes WWW rozhraní (protokoly HTTP, SOAP)
 - postavené na XML
 - Komponenty pro tvorbu distribuovaných aplikací v prostředí internetu.
- Publikace komponent
 - WSDL – Web Services Description Language
- Vyhledávání komponent
 - UDDI – Universal Description, Discovery and Integration
- Více v předmětu *Vývoj internetových aplikací*

Komponenty – Webové služby (2)





Komponenty – Rozdíly mezi OOP

- OOP se zaměřuje na vztahy mezi třídami, které jsou do jedné spustitelné aplikace (*white box programming*).
- Skládání aplikace z nezávisle pracujících bloků (*black box programming*).
 - Oddělený vývoj aplikace a komponent ze kterých je vytvářena
 - Musí podporovat systém verzí.
 - Oddělení rozhraní od implementace.

Komponenty – Hlavní komponentní technologie



- JavaBeans, EJB
 - pouze jazyk Java
 - Komponenty uživatelského rozhraní
 - Enterprise Java Beans – pro rozsáhlé systémy
- COM, COM+, DCOM, ActiveX
 - Binárně kompatibilní komponenty
 - Základní technologie pro Windows
- .NET
 - Kompatibilita na jazykové úrovni – C++, C#, Jscript, VB.NET

Java Beans



- “Write once, run anywhere, reuse everywhere”
 - Přidávání funkcí bez nutnosti přepisovat úplně všechno
 - Provádění na všech platformách
 - Použití v různých scénářích – aplikacích, jiných komponentách, dokumentech, www stránkách, nástrojích pro vývoj aplikací, ...
- Komponenta jako stavební blok
 - Kontejnery – kombinace komponent do struktur
 - Principy manipulace a komunikace s komponentami z vnějšího prostředí
 - Introspekce
 - Zpracování událostí
 - Persistence

Java Beans – Typy komponent



- Vizuální komponenty
 - Mají vizuální reprezentaci zabírající prostor v okně aplikace
 - Příklad: tlačítko, tabulka, rolovací seznamy
 - Podpora ve vizuálních nástrojích
- Nevizuální komponenty
 - Příklad: časovač, databázové spojení, kontrola pravopisu, ...

Java Beans – Scénáře použití (1)



- Využití grafického návrhového prostředí
 - Rozložení komponent v okně aplikace
 - Nastavení vlastností komponent
 - Barva, typ písma, klávesové zkratky, ...
 - Využití editorů vlastností (property editor)
 - Propojení komponent a zápis metod pro zpracování událostí
 - Testování
 - Vytvoření instalace aplikace včetně komponent

Java Beans – Scénáře použití

(2)



The screenshot displays the JBuilder 2006 IDE interface. The main workspace shows a Swing window with two components: a `JTextField` and a `JButton`. The Properties palette on the right is open, showing the configuration for the selected `JTextField`.

Property	Value
name	JTextField1
constraints	
actionMap	
alignmentX	0.5
alignmentY	0.5
background	White
border	Line
caretColor	Orange
caretPosition	11
columns	0
debugGraphicsOpti...	<default>
disabledTextColor	172, 168, 153
document	
doubleBuffered	False
editable	True
enabled	True
FocusAccelerator	
font	"Tahoma", Font.PLAIN
foreground	Black
horizontalAlignment	LEADING
inputVerifier	
margin	1, 5, 2, 4
maximumSize	2147483647, 2147483647
minimumSize	11, 19
nextFocusableCom...	
opaque	True
preferredSize	64, 19
requestFocusEnabled	True
scrollOffset	1
selectedTextColor	Black
selectionColor	178, 180, 191
selectionEnd	11
selectionStart	11
text	JTextField1
toolTipText	
verifyInputWhenFo...	True

Java Beans – Scénáře použití

(3)



- Použití v ručně psaném programu
 - Vytvoření instancí komponent a nastavení jejich rozměrů a pozic
 - Nastavení vlastností komponent
 - Vytvoření metod pro zpracování událostí
 - Registrace metod pro zpracování událostí
 - Testování
 - Vytvoření instalace aplikace včetně komponent

Java Beans – Scénáře použití (4)



```
private void jbInit() throws Exception {
    jTextField1.setCaretColor(Color.orange);
    jTextField1.setText("jTextField1");
    jButton1.setText("jButton1");
    jButton1.addActionListener(
        new JbExample_jButton1_actionAdapter(this));
    this.add(jTextField1);
    this.add(jButton1);
}

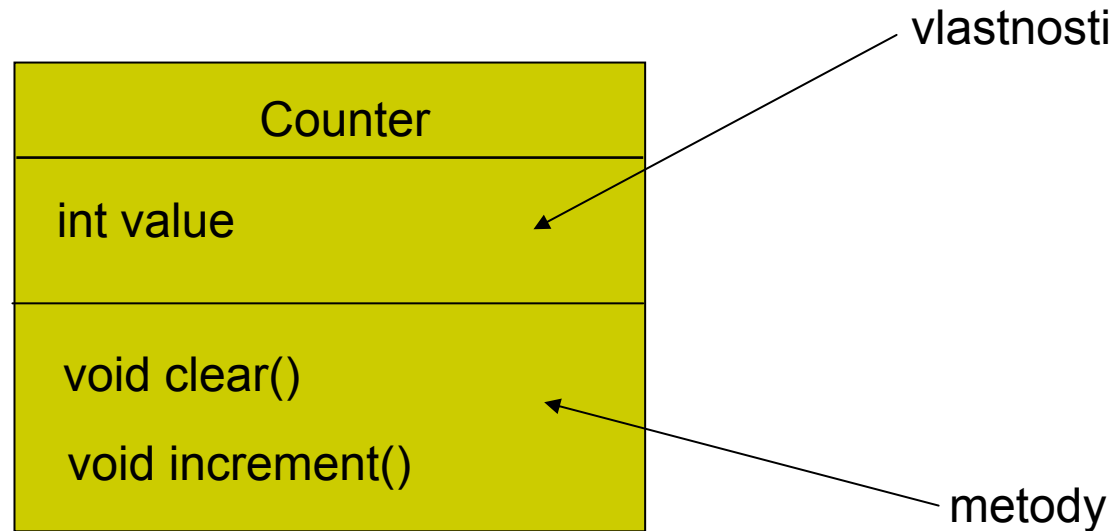
JTextField jTextField1 = new JTextField();
JButton jButton1 = new JButton();
public void jButton1_actionPerformed(ActionEvent e) {
    //TODO: put your code here
}
```

Java Beans – Struktura komponenty

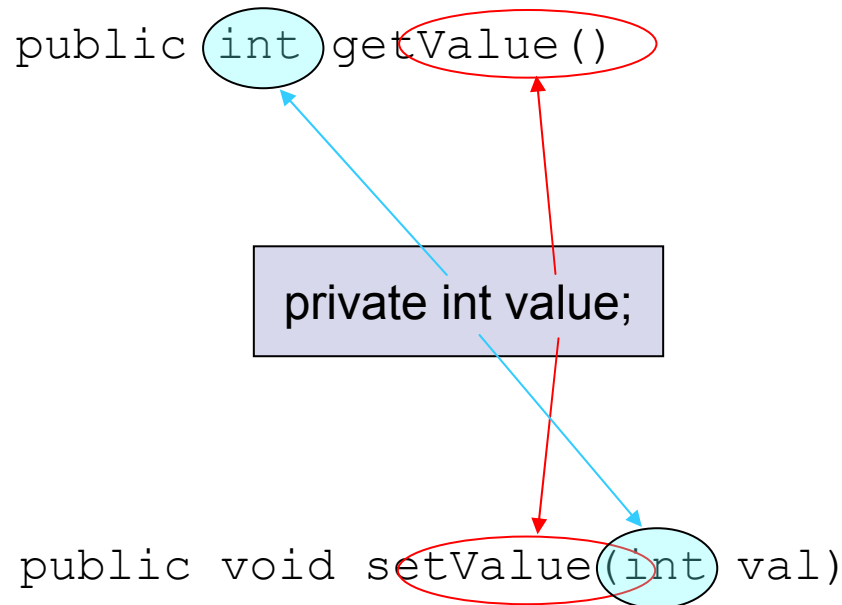


- Vlastnosti (properties)
 - Přístup prostřednictvím *přístupových metod* (čtení, zápis hodnoty), ne přímo
- Metody (methods)
 - Operace nad komponentami
- Události (events)
 - Vazby mezi komponentami

Java Beans – Příklad komponenty *Counter*



Java Beans – Přístupové metody (1)



Java Beans – Přístupové metody (2)



```
public class Counter {  
    private int value;  
    public int getValue()  
        return this.value;  
}  
public void setValue(int val) {  
    this.value=val;  
}  
...  
}
```

Java Beans – Přístupové metody (3)



- Vlastnosti určené pouze pro čtení
 - Mají pouze metodu `getXXX()`
- Vlastnosti určené pouze pro zápis
 - Mají pouze metodu `setXXX()`
- Vlastnosti typu boolean
 - Metoda pro čtení se může jmenovat `isXXX()`
 - `public boolean isEmpty();`

Java Beans – Indexované vlastnosti



```
public int getValue(int index)
```

```
private int[] value;
```

```
public void setValue(int index, int val)
```

Java Beans – Speciální vlastnosti



- Vázané vlastnosti (bound properties)
 - Generují událost `PropertyChange`, pokud se mění jejich hodnota
- Vlastnosti s omezením (constrained prop.)
 - Generují událost `VetoableChange`, pokud se mění jejich hodnota
 - Změna může být zakázaná

Java Beans – Použití vlastností komponent



- Atributy objektů ve skriptovacích jazycích
 - JavaScript, VBScript
- Programový přístup přes veřejné přístupové metody
- Přístup přes formuláře (property sheets) v návrhových prostředích
- Čtení a zápis do perzistentní paměti



Java Beans – Metody

- Za metody komponenty se považují všechny veřejné (public) metody třídy

```
public void clear() {  
    val=0;  
}
```

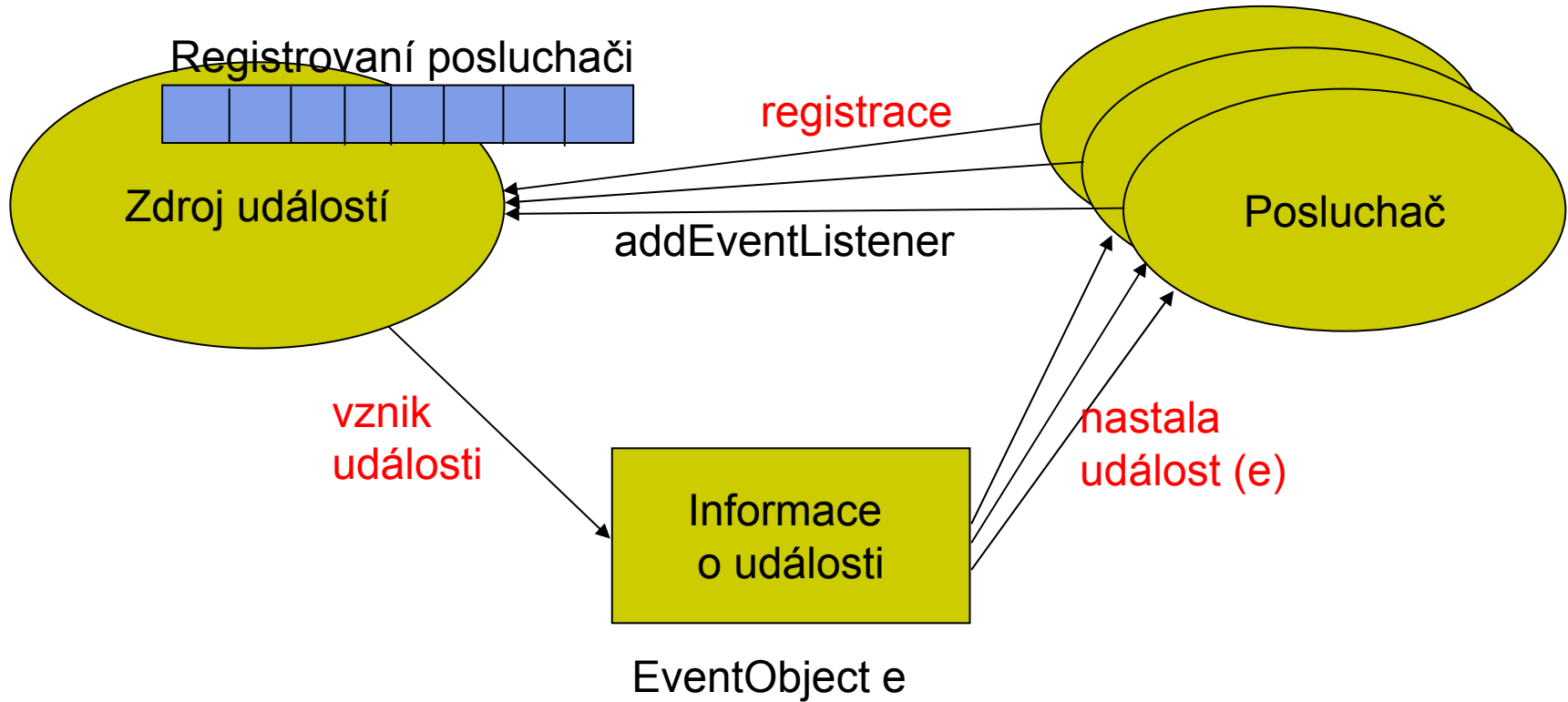
```
public void increment() {  
    val++;  
}
```



Java Beans – Události (1)

- Zdroj událostí
 - Objekt, který generuje události
 - Spravuje seznam registrovaných posluchačů
- Posluchač (listener)
 - Objekt, který chce být o události informován
 - Musí být registrován u zdroje událostí
 - Musí implementovat dohodnuté rozhraní

Java Beans – Události (2)



Java Beans – Zpracování událostí



1. Posluchač se zaregistruje u zdroje událostí (např. u tlačítka, na jehož stisknutí čeká)
2. Uživatel stiskne tlačítko – vznikne událost
3. Zdroj události (tlačítko) projde seznamem registrovaných posluchačů a každému z nich oznámí vznik události:
 - Zavolá dohodnutou metodu rozhraní posluchače
 - Metodě předá informace o události (podtřída `java.util.EventObject`)

Java Beans – Informace o události



```
//Objekt přenášející informace i vygenerované
//události
public class CounterEvent extends EventObject {

    public CounterEvent(Counter source) {
        super(source);
    }

    public Counter getCounter() {
        return (Counter)source;
    }

}
```




Java Beans – Posluchač (1)

```
public interface CounterEventListener
    extends EventListener
{
    //byla překročena maximální hodnota
    void limitReached(
        CounterEvent event);

    //čítač byl vynulován
    void reset(CounterEvent e);
}
```

Java Beans – Posluchač (2)



```
class MyListener
    implements CounterEventListener
{
    public void limitReached(CounterEvent event) {
        System.out.println("Čítač vynulován");
    }
    public void reset(CounterEvent e) {
        System.out.println("Překročen limit!");
    }
    ...
}
```

Java Beans – Registrace posluchače (1)



```
public class Counter {  
  
    ...  
  
    public void addCounterEventListener(  
        CounterEventListener listener)  
    {  
        //registrace posluchačů  
    }  
    public void removeCounterEventListener(  
        CounterEventListener listener)  
    {  
        //zrušení registrace posluchače  
    }  
}
```

Java Beans – Registrace posluchače (2)



```
class MyListener
    implements CounterEventListener {

    void run() {
        Counter counter = new Counter();
        counter.addCounterListener(this);
        counter.clear();

        ...
    }

    ...
    public void limitReached(CounterEvent event) {...}
    public void reset(CounterEvent e) {...}

}
```



Java Beans – Adaptér (1)

- Rozhraní `EventListener` pro konkrétní komponentu může obsahovat mnoho metod
- Chceme-li reagovat jen na některé události:
 1. Musíme buď implementovat prázdné reakce na ostatní události
 2. Nebo použijeme *adaptér* jako bázovou třídu a implementujeme jen zvolené metody
- *Adaptér* implementuje implicitní odezvu na všechny události



Java Beans – Adaptér (2)

```
class CounterAdapter
    implements CounterEventListener {

    public void reset(CounterEvent e)
    {}

    public void limitReached(
        CounterEvent e)
    {}

}
```



Java Beans – Adaptér (3)

```
Counter counter = new Counter();

//použití anonymní vnitřní třídy
counter.addCounterListener(
    new CounterAdapter {
        public void reset(CounterEvent e) {
            System.out.println("Reset");
        }
    }
);
```

Java Beans – Enterprise Java Beans (EJB)



- *Specifikace architektury pro vývoj a nasazení distribuovaných transakčních objektových komponent na straně serveru*
- Konvence + sada rozhraní (EJB API)
- Cíl = zajištění kompatibility mezi produkty různých výrobců
 - komponenty
 - kontejner



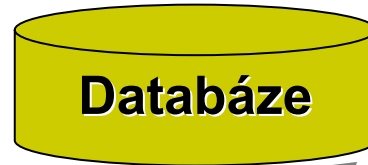
Java Beans – EJB kontejner

- prostředí, v němž běží komponenty
 - vzdálený přístup
 - bezpečnost
 - transakce
 - souběžný přístup
 - přístup ke zdrojům a jejich sdílení
- izolace komponent od aplikací
 - nezávislost na dodavateli kontejneru
 - zjednodušení tvorby aplikací

Java Beans – Aplikační komponenty v J2EE



datová
vrstva



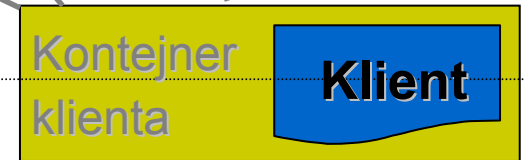
aplikační
logika



prezentační
logika



prezentace



Java Beans – Typy komponent

EJB



- Session Beans
 - bezstavové služby
 - stavové objekty v rámci sezení
- Entity Beans
 - perzistentní objekty spravované uživatelem (BMP)
 - perzistentní objekty spravované kontejnerem (CMP)
- Message-Driven Beans
 - bezstavové služby volané asynchronně

Java Beans – Další informace



- Využívají se např. v informačních systémech
- Odkazy na domovské stránky k Java Beans
 - <http://java.sun.com/products/javabeans/>
- Více informací o technologii EJB bude v předmětu Tvorba informačních systémů.



COM – Historie (1)

- 1990 – OLE 1.0 (Object Linking and Embedding) – strukturované dokumenty
 - Příklad: vložení grafu z Excelu do Wordu
- 1991 – Visual Basic 1.0 - VBX
- 1993 – OLE 2.0, COM (Component Object Model)
 - Použito ve Windows 3.1, základ pro Windows 95
- 1994 – OLE controls (OCX)
- 1996 – ActiveX, DCOM (Distributed COM)



COM – History (2)

- 2000 – COM+ (Windows 2000)
 - Distribuované komponenty
 - „Farmy komponent“ – MTS (MS Transaction Server)
- 2002 – .NET Framework
 - Zachovaná návaznost na COM+



COM – Aplikace ve Windows

- Každá aplikace má oddělený adresový prostor.
 - Virtuální paměť
 - Překlad logických adres na fyzické.
 - Běh více aplikací
- Ve Windows máme možnost použít:
 - Samostatné aplikace - .EXE soubory
 - Dynamicky linkované knihovny - .DLL soubory
 - Obsahují spustitelný kód, nedají se ovšem spustit přímo.

COM – Component Object Model (1)



- COM je specifikace pro tvorbu komponent.
- COM je technologie, která implementuje část této specifikace.
- COM je postaven na třech základních principech.
 - Při vytváření aplikace programátoři využívají rozhraní.
 - Zdrojový kód komponenty není staticky připojen, spíše nahrán na požádání za běhu aplikace.
 - Programátoři implementující COM objekty deklarují požadavky a systém zajistí, že tyto požadavky budou splněny.

COM – Component Object Model (2)



- **COM rozhraní** je kolekce abstraktních operací, které mohou být provedeny objektem.
 - Musí rozšiřovat rozhraní IUnknown
 - Identifikován UUID – IID (použití GUID)
 - Z hlediska implementace jde o abstraktní třídu s čistě virtuálními metodami.
 - Jazyk pro popis rozhraní je (M)IDL
- **COM třídy**
 - konkrétní implementace jednoho nebo více rozhraní.
 - Nelze ztotožňovat například s třídou v jazyce C++ (třída v C++ může být COM třídou...)
 - Identifikován UUID – CLSID (použití GUID)

COM – Component Object Model (3)



- **COM Objekt**

- je instancí nějaké třídy
- COM objekty jsou registrovány v databázi komponent
- Nemají identitu.

- **Komponenta**

- Binární soubor obsahující výkonný kód.
- Jedna nebo více zapouzdřených jednotek (COM tříd), které mohou využívat klienti.

COM – Global component and interface ID



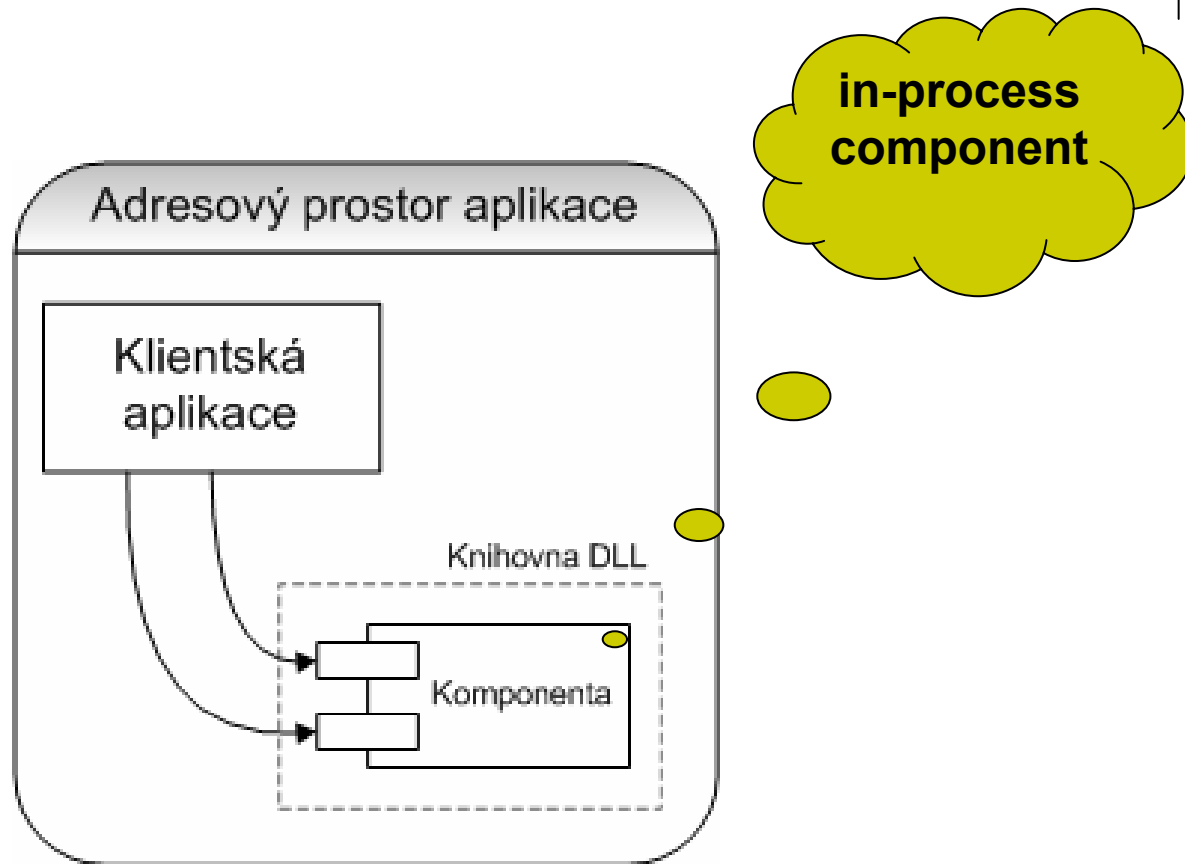
- Globální identifikace komponent a jejich rozhraní (Global component and interface ID - GUID)
 - 16 byte – 10^{19} kombinací)
 - {3F2504E0-4F89-11D3-9A0C-0305E82C3301}
 - Algoritmus již nezahrnuje MAC adresu (bylo možné vystopovat autora dokumentu – červ Melissa)
 - Utility přímo v systému GUIDGEN.EXE (generátor součástí například Visual Studia).
 - Obvykle se používá jako konstanta
 - `const IID IID_IRandom = {0x771853E0,0x78D1,0x11d7,{0xBF,0xB4,0xED,0x72,0x61,0xDE,0xA8,0x3D}};`



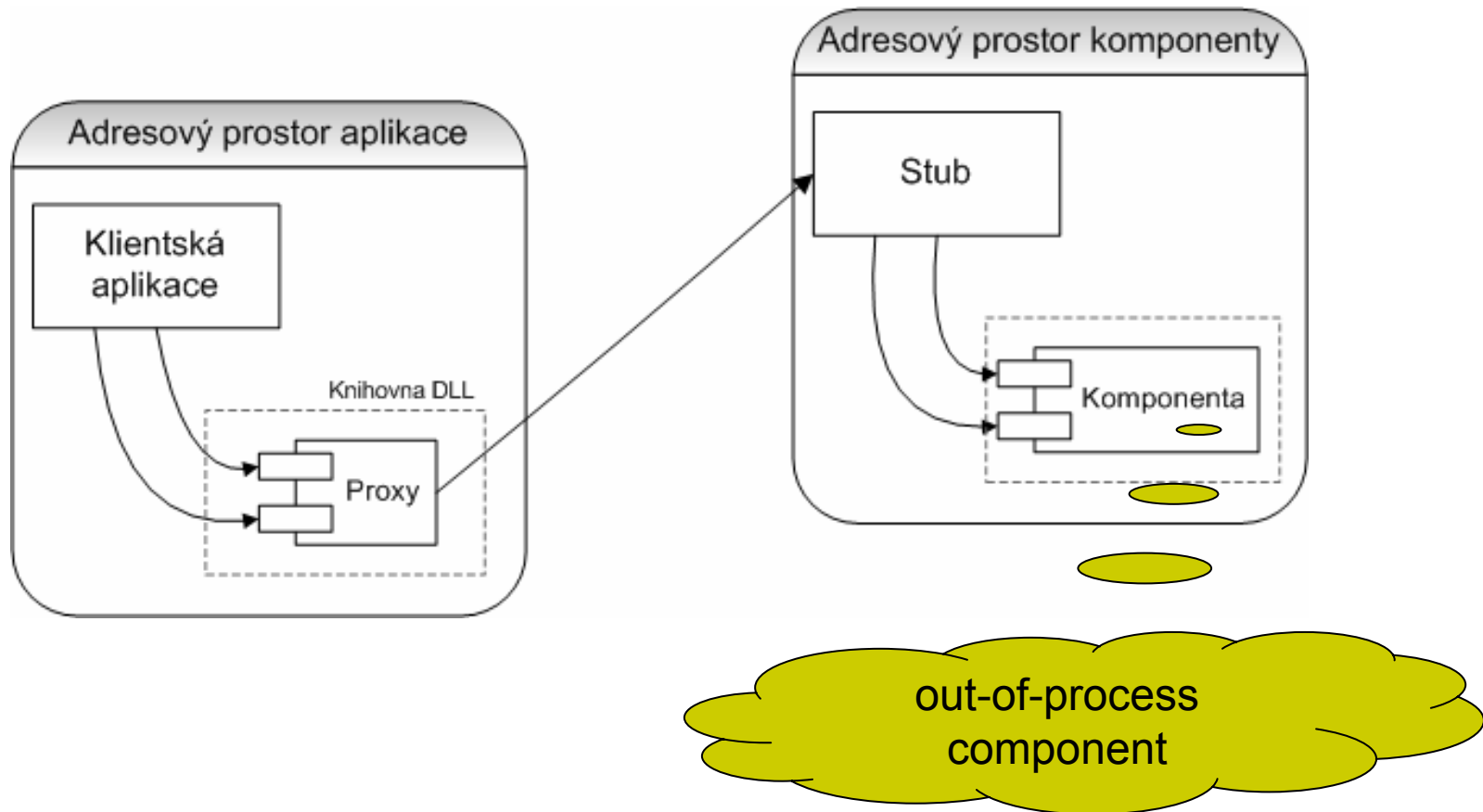
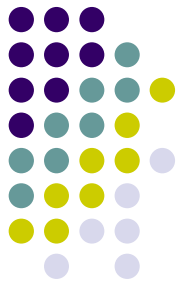
COM – Typy komponent

- **in-process**
 - běží ve stejném adresovém prostoru a na stejném procesoru jako aplikace (.DLL)
- **out-of-process**
 - běží v samostatném adresovém prostoru nebo na samostatném serveru(.EXE)
 - vyžadují zajištění přenosů požadavků a výsledků (marshalling, serialization)
 - komunikace s komponentou prostřednictvím zástupce (proxy, stub)
- COM umožňuje použití komponent na jednom počítači, ale v rámci různých adresových prostorů
- DCOM – Distributed COM

COM – Komunikace s komponentami (in-process)



COM – Komunikace s komponentami (out-of-process)

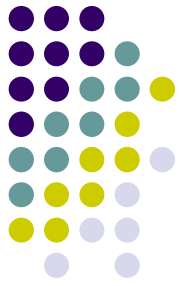


COM – Rozhraní IUnknown



- Rozhraní identifikováno IID.
- Jak získat přístup ke konkrétnímu rozhraní?
 - Každá COM komponenta implementuje rozhraní IUnknown (přímo nebo nepřímo)
- Metody:
 - QueryInterface – vrátí adresu implementace rozhraní se zadaným GUID
 - AddRef – zvýší počítadlo referencí
 - Release – sníží počítadlo a případně může uvolnit s paměti

COM – Příklad komponenty (1)



```
// File IRandom.h
#pragma once
#include "unknwn.h"

// {CB8DF8CB-3F6C-4c02-A587-18566C28487B}
DEFINE_GUID(IID_IRandom, 0xcb8df8cb,
0x3f6c, 0x4c02, 0xa5, 0x87, 0x18, 0x56, 0x6c, 0x28, 0x48, 0x7b) ;

interface IRandom : IUnknown {
    virtual HRESULT STDMETHODCALLTYPE Start(
        int seed) = 0;
    virtual HRESULT STDMETHODCALLTYPE Next(
        int *val) =0;
};
```


COM – Příklad komponenty (2)



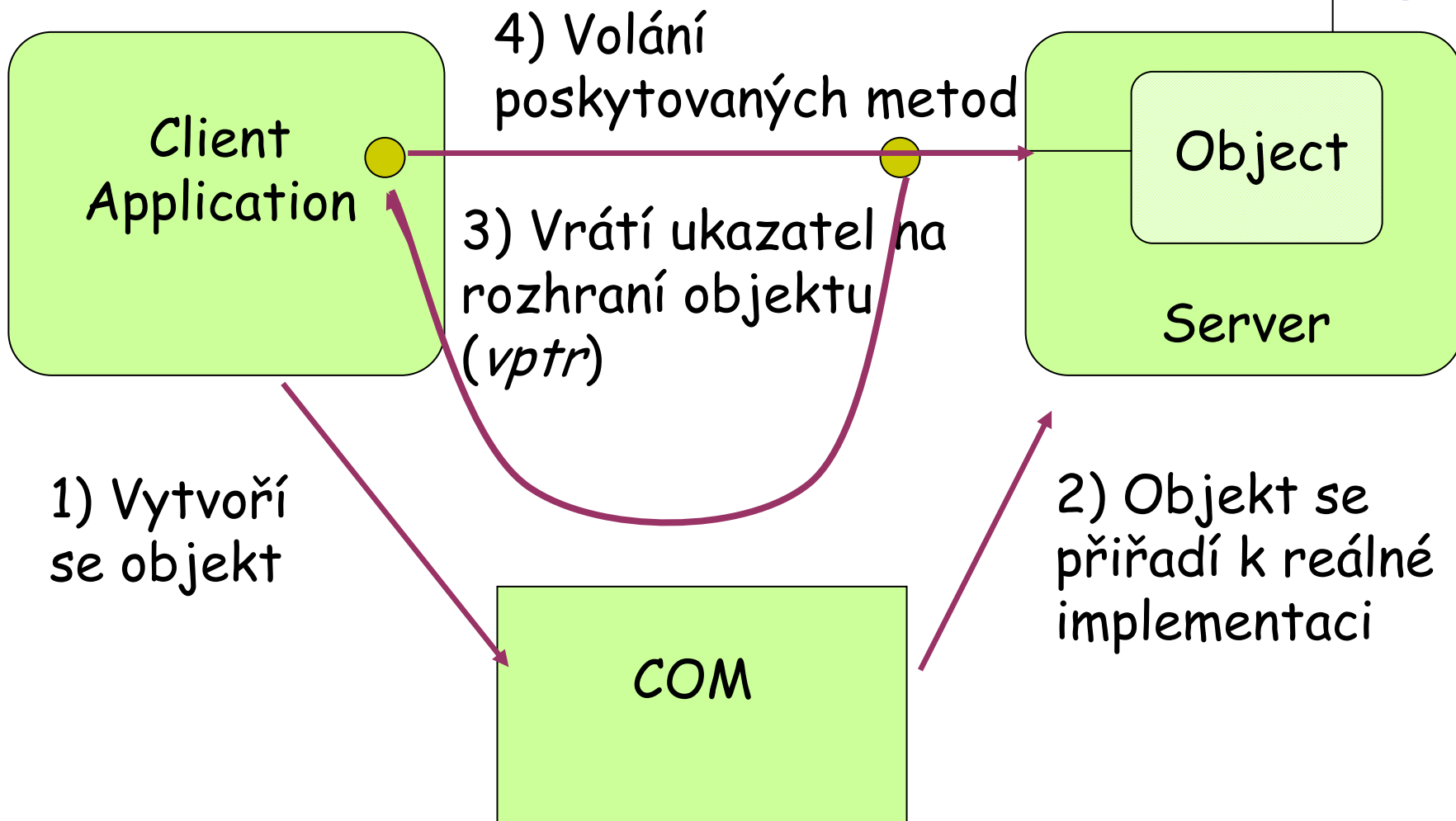
```
#pragma once
#include "IRandom.h,,

class CRandomImpl : public IRandom {
public:
    // IUnknown
    STDMETHODIMP QueryInterface(REFIID, void **);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

    // IRandom
    STDMETHODIMP Start(int seed);
    STDMETHODIMP Next(int* result);

private:
    ULONG m_refCnt;
    ULONG m_seed;
};
```

COM – Použití COM komponenty (1)



COM – Použití COM komponenty (2)



```
#define INITGUID
#include "IRandom.h"
#include <iostream>
using namespace std;

void main(int argc, char** argv) {
    // initialize COM
    HRESULT hr = CoInitialize(NULL);
    if (FAILED(hr)) { ... }
    // create component instance
    IRandom *pRnd = NULL;
    hr = CoCreateInstance(CLSID_RandomImpl, NULL,
        CLSCTX_ALL, IID_IRandom, (void **)&pRnd);
    if (FAILED(hr)) { ... }
    // use the component
    pRnd->Start(1234);
    // release the komponent
    pRnd->Release();
    // COM finalization
    CoUninitialize();
}
```

COM – Použití COM komponenty (3)



1. Je nutné inicializovat systém funkcí **CoInitialize (COMPOBJ.DLL)**.
2. Je zavolána funkce **CoCreateInstance**. Jako argument je předán CLSID použitého objektu. Funkce:
 - Použije registry k tomu, abychom identifikovali komponentu, kterou chceme vytvořit.
 - Je vytvořena požadovaná instance komponenty.
 - Třetí argument určuje, kde se komponenta vykonává (ve stejném adresovém prostoru, v jiném, na jiném počítači).
 - Je volána ClassFactory (jde i jinak, získat příslušný „factory“ objekt a volat přímo).
 - Obdrží ukazatel na požadované rozhraní.

COM – Použití COM komponenty (4)



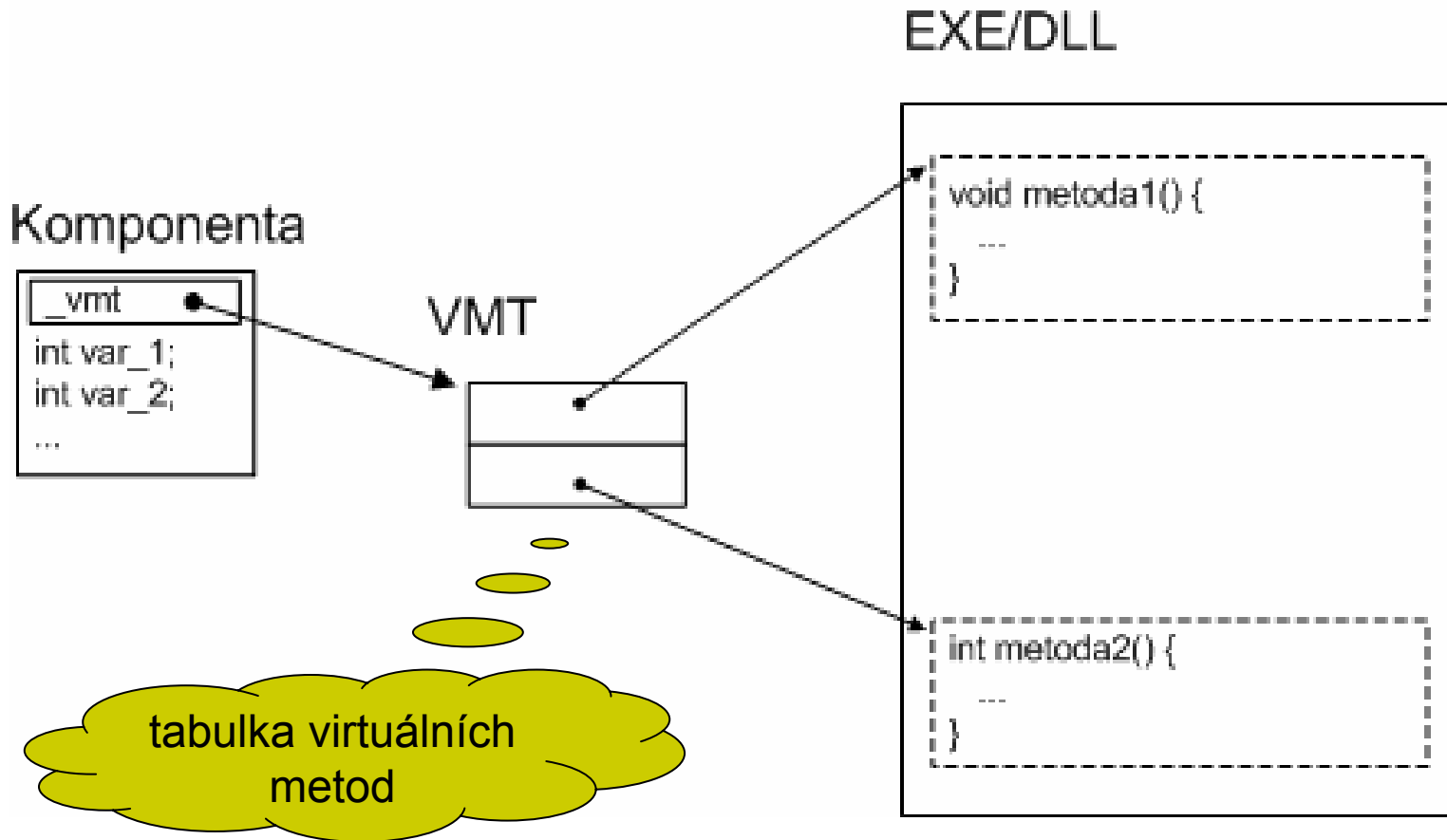
3. Získané rozhraní je určité typu IUnknown.
 - Můžeme požit QueryInterface k přístupu k dalším rozhráním.
 - `HRESULT QueryInterface(REFIID riid, void**ppv);`
4. Použijeme metody Addref a Release k řízení životního cyklu komponenty.
5. Můžeme volat metody komponenty.
6. Ukončíme voláním **CoUnitalize**.

COM – Binární kompatibilita



- COM komponenty jsou kompatibilní na binární úrovni.
- Standardní způsob jak:
 - uložit v paměti virtuální tabulku funkcí (*vtable*);
 - volat funkce prostřednictvím *vtable*.
- Každý jazyk (C, C++, Small Talk, Ada, and dokonce Basic) který může volat funkce prostřednictvím ukazatelů, může být využit k realizaci komponent.
- Ty pak mohou spolupracovat s ostatními komponentami, které používají stejný binární standard.

COM – Tabulka virtuálních metod



COM – Znovupoužití komponent



- Component containment
 - komponenta *používá* nějakou vnitřní komponentu
 - Obě mají vlastní implementaci `IUnknown`
- Component aggregation
 - Komponenta *zpřístupňuje* nějakou vnitřní komponentu
 - Obě komponenty sdílí jednu implementaci `IUnknown`



COM – COM+ (1)

- Zachovává vlastnosti COM/DCOM. Navíc přináší:
 - Konfigurované komponenty
 - Aplikační služby
 - Automatické transakce- možnost nastavit konfigurací
 - Just-in-time aktivace – je využít *kontextový* objekt
 - Object pooling – možnost odložit nepotřebný objekt, uvolnit jej z paměti (stav je uložen do perzistentní paměti)
 - Fronty volání
 - Události

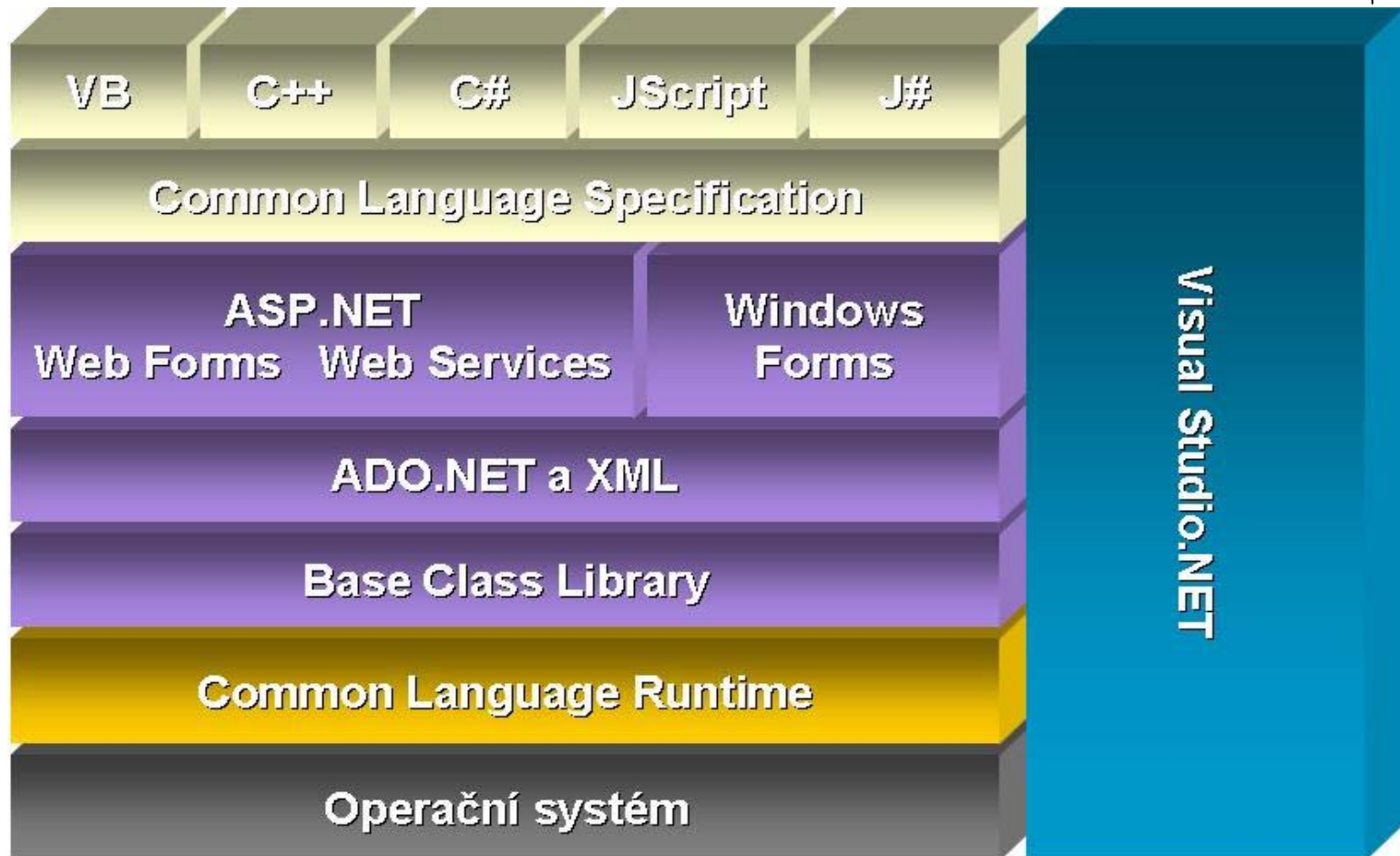
COM – COM+ (2)



- Konfigurované komponenty
 - Krom informací v registry přidán i **katalog**.
 - Můžeme nastavovat vlastnosti komponenty.
 - Například které aplikační služby má podporovat.
 - Konfigurovaná komponenta je ta, která má záznam v katalogu...
- Zaveden nový pojem **kontext**.
 - Kontext je seskupení objektů se shodně nakonfigurovanými vlastnostmi.
 - V rámci jednoho kontextu spolu mohou komunikovat COM objekty přímo, jinak musí použít *proxy*.
 - Kontextový objekt
 - Objekt reprezentující daný kontext.



.NET - Architektura .NET

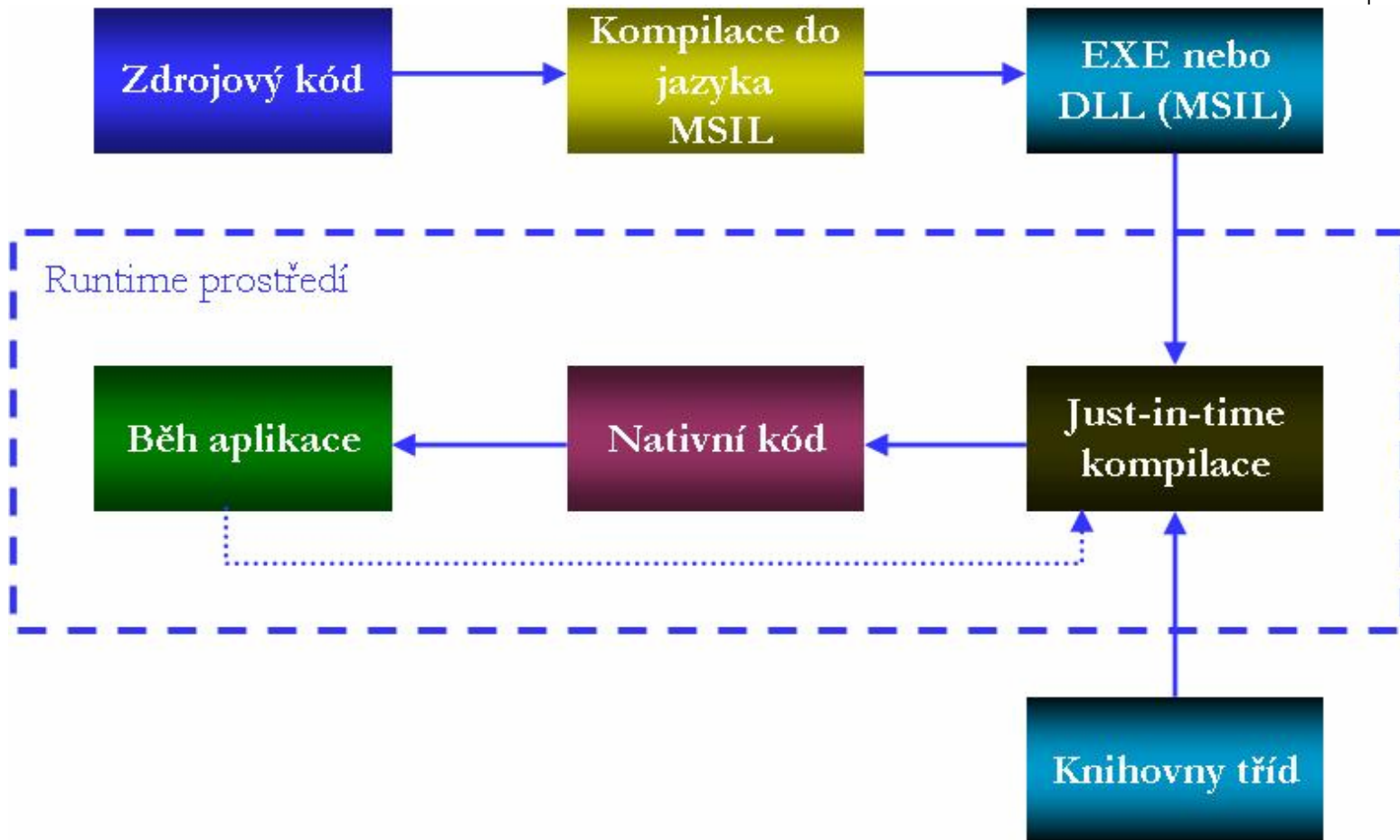


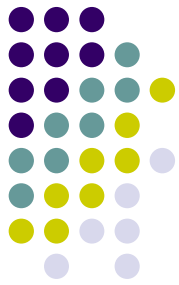
.NET - Součástí .NET Framework



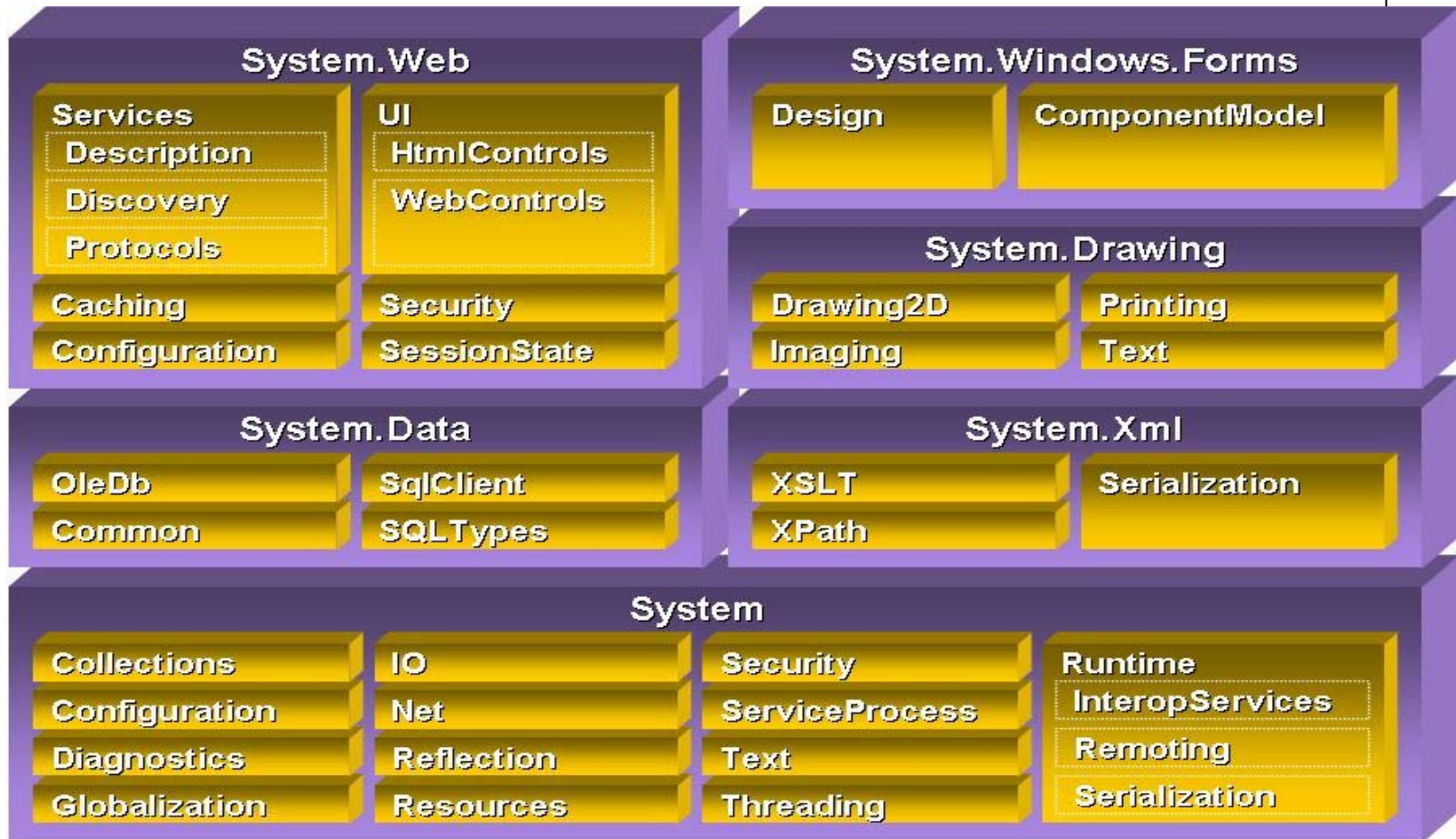
- **CLR** – společné běhové prostředí
- **ADO.NET** – přístup k datům
- **ASP.NET** – webové aplikace
- **Windows Forms** – uživatelské rozhraní
- **CLS** – společné vlastnosti jazyků
 - CTS – společné typové prostředí
- **Programovací jazyky** – C#,...

.NET - Kompilace v prostředí .NET





.NET - Knihovny .NET





.NET - Historie

- 1998 – první zprávy o novém jazyce COOL
- červen 2000 – uvolnění specifikace C#
- červenec 2000 - .NET Framework SDK
- listopad 2000 – zahájení standardizace
- 13. 12. 2001 – ECMA-334 (C#), 335 (CLI)
- leden 2002 – finální podoba .NET
- Dnes verze 2.x připravuje se 3.0

- Autoři: Anders Hejlsberg (Turbo Pascal, Delphi), Scott Wiltamuth, Peter Golde



C# - Vlastnosti jazyka C# (1)

- Podpora vývoje založeného na komponentách
 - přímá podpora základních konstrukcí (vlastnosti, metody, události)
 - metadata + atributy
 - integrované verzování komponent
 - integrovaná XML dokumentace



C# - Vlastnosti jazyka C# (2)

- Moderní objektově orientovaný jazyk
 - veškeré očekávané vlastnosti
 - delegát – bezpečné předávání funkcí jako parametrů
 - přetěžování operátorů, uživatelské konverze, „obyčejná“ pole
 - rozsáhlý systém knihoven tříd
 - jednotný typový systém
 - hodnotové – referenční typy (boxing / unboxing)



C# - Vlastnosti jazyka C# (3)

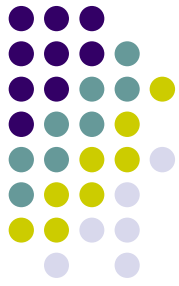
- Vývoj robustních a trvanlivých aplikací
 - typová bezpečnost
 - automatická správa paměti
 - jednotný systém výjimek
 - možnost nezávislého verzování bazových tříd
 - kontroly rozsahů polí, aritmetické kontroly



C# - Vlastnosti jazyka C# (4)

- Pragmatický přístup
 - žádná „revoluce“ v syntaxi jazyka
 - práce s ukazateli, přetypování, přímé přidělování paměti na zásobníku – nutnost speciálních privilegií
 - spolupráce s existujícími komponentami COM, DLL

C# - Příklad



```
using System;

namespace Prikklady
{
    /// <summary>
    /// Summary description for Prikklad
    /// </summary>
    class Prikklad
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



C# - Datové typy (1)

- Hodnotové typy
 - obsahují přímo data – přiřazením vzniká kopie hodnoty
 - sbyte, byte - 10, 3u, 0x00ff
 - short, ushort, int, uint, long, ulong - 1000L
 - char – 'a', '\x0041', '\n'
 - float, double, decimal - 1.5f, 1e-3m, 6.67e-23d
 - bool – true, false



C# - Datové typy (2)

- struktura (struct)
 - Hodnotový typ
 - Jako třída, ale bez dědičnosti

```
struct Komplex {  
    public float real;  
    public float imag;  
}
```



C# - Datové typy (3)

- Výčtový typ
 - množina pojmenovaných konstant

```
enum DnyTydne {  
    Pondeli, Utery, Stredam ... };
```

```
enum Dny : byte { ...};  
Dny dnes = Dny.Nedele;
```



C# - Datové typy (4)

- Referenční typy
 - neuchovávají hodnotu, ale odkaz na hodnotu
 - hodnota *null*
 - přiřazením nevzniká kopie hodnoty, ale jen odkazu
 - Typy object, string, třída (class), rozhraní(interface), pole a delegát(delegate)
- Ukazatele



C# - Datové typy (5)

- Pravidelná pole
 - Klasická pole jako v C/C++
 - Stejný počet prvků ve všech řádcích
 - `int [,] mat1 = new int[5,10];`
 - `mat1[2,3] = 15;`
- Nepravidelná pole
 - Pole polí jako v Javě
 - `int[][] mat2 = new int [5][];`
 - `mat2[2] = new int[7];`
 - `mat[2][3] = 15;`



C# - Výrazy a příkazy

- Jako C/C++/Java
 - `Console.WriteLine("ahoj "+Priklad.f(5*6));`
- Příkaz `switch` – nelze přecházet mezi větvemi
- Příkaz `goto`
- Průchod kolekcí nebo polem – příkaz `foreach`
`foreach(T x in IEnumerable) {...}`

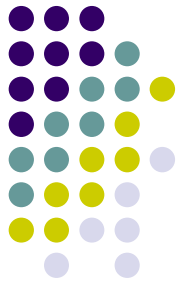
```
IEnumerable ie=coll.GetEnumerator();  
while( ie.MoveNext() ) {  
    T x=(T)ie.Current();  
    ...  
}
```



C# - Operátory

- Primární - (x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked
- Unární - + - ! ~ ++x --x (T)x
- Multiplikativní - * / %
- Aditivní - + -
- Posuvu - << >>
- Relační - < > <= >= is as
- Rovnosti - == !=
- Logické bitové - & | ^
- Logické spojky - && ||
- Podmínečné - ?:
- Přiřazovací - = += -= *= /= %= &= |= ^= <<= >>=

```
if (someObject is SomeInterface)
{
    return someObject;
} else {
    return null;
}
```



C# - Jmenné prostory

```
using SomeNamesapce;  
using AnotherNamesapce;  
namespace A {  
    class Class1  
    {...}  
    class Class2  
    {...}  
}  
namespace B {  
    namespace C  
    {...}  
    class Class1  
    {...}  
}
```



C# - Třídy a rozhraní

- Obsahuje: položky; metody; vlastnosti; indexery; operátory; události.
- Modifikátory přístupu: `private`, `protected`, `internal`, `public`
- Další modifikátory: `static`, `sealed`, `abstract`
- `class`, `interface`
 - `IName` - konvence pro pojmenování rozhraní

C# - Třídý



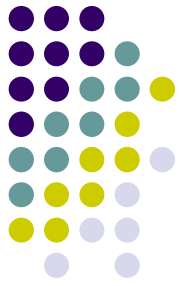
```
class Point {  
    private short x, y;    //data  
    public short X {      //vlastnost  
        get { return x; }  
        set { x = value; }  
    }  
    public short GetY {   //metoda  
        return y;  
    }  
}
```

C# - Indexery



```
class SetOfPoints {
    Point[] points;
    public SetOfPoints(int size) {
        points=new Point[size];
    }
    public Point this[int index] {
        set {
            if (index<points.Length) points[index]=value;
        }
        get {
            if (index<=points.Length) return points[index];
            else return new Point(0,0);
        }
    }
}
```

C# - Přetěžování operátorů



```
class Test
{
    protected string text;
    public string Text
    {
        get { return this.text; }
        set { this.text=value; }
    }
    public static Test operator + (Test t1,Test t2)
    {
        return new Test(t1.Text+" "+t2.Text);
    }
}
```


C# - Dědičnost a polymorfismus



```
class A {  
    public void SomeMethod() { }  
    public virtual void AnotherMethod() { }  
}  
class B : A {  
    public new void SomeMethod()  
    { //původní metoda je překryta }  
    public override void AnotherMethod() { }  
}
```

```
A a=new B();
```

```
a.SomeMethod(); //zpustí původní metodu třídy A
```

```
a.AnotherMethod(); //zpustí metodu třídy B
```

C# - konstruktory a destruktory



- Stejné jméno jako třída
 - `public Bod(int x, int y) {x=0; y=0;}`
- Lze přetěžovat
 - `public SomeName(...) :base (...) { ... }`
- Statické konstruktory
 - Není zaručeno pořadí volání
 - `static A() { ...}`
- Konstruktor se volá automaticky
- Destruktor se volá při zrušení objektů – garbage collector



C# - Výjimky

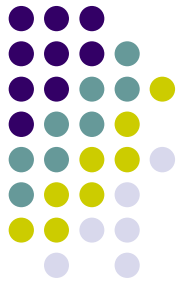
```
try
{ ... }
catch (System.FileException e)
{ ... }
catch (System.Exception e)
{ ... }
finally
{ ... }
```



C# - Delegáti (1)

- Typově bezpečný ukazatel na funkci
 - call-back metody
- Delegáti se používají jako metody pro událostní programování
- Deklarace:
 - `modifiers delegate type delegatsName (parameter1, ...);`
- Před použitím je potřeba delegáta instanciovat – svážeme jej s konkrétní funkcí

C# - Delegáti (2)



```
class Text
{
    string text;
    public Text(string text)
    {
        this.text=text;
    }
    public delegate void SomePrefix();
    public void WriteIt(SomePrefix prefix)
    {
        prefix(); //jako běžná metoda
        Console.WriteLine(text);
    }
}
```



C# - Delegáti (3)

```
class PrefixBuilder
{
    public static void SimplePrefix()
    {
        Console.WriteLine("## ");
    }
    public void NicePrefix() {
        Console.WriteLine(">-how nice-> ");
    }
}
```

C# - Delegáti (4)



```
class RunApp {
    static void Main() {
        Text text=new Text("Hello");

        Text.SomePrefix simplePrefix=new
        Text.SomePrefix(PrefixBuilder.SimplePrefix);
        PrefixBuilder prefixBuilder=new PrefixBuilder();
        Text.SomePrefix nicePrefix=new
            Text.SomePrefix(prefixBuilder.NicePrefix);
        text.WriteIt(simplePrefix);
        text.WriteIt(nicePrefix);
    }
}
## Hello
>-how nice-> Hello
```



C# - Kompozice delegátů

- Delegát nemusí být interně spojen jen s jednou metodou
- kompozici provedeme pomocí operátorů + a –.

```
Text.SomePrefix greatPrefix=simplePrefix +  
    nicePrefix + simplePrefix;
```

```
text.WriteIt(greatPrefix);
```

```
greatPrefix-=nicePrefix;  
text.WriteIt(greatPrefix);
```

- Výstup programu bude:
>-how nice-> ## Hello
Hello



C# - Události (1)

- Zpracování událostí realizováno pomocí delegátů
 - Delegát musí mít dva parametry a oba jsou objekty
 - První udává kdo je zdrojem události
 - Druhý obsahuje informace spojené s konkrétní událostí - třída *EventArgs*
- Definice události je součástí třídy:
 - `event JmenoDelegata JmenoUdalosti;`

C# - Události (2)



```
class InfoEventArgs : EventArgs
{
    private string info;
    public InfoEventArgs(string info)
    {
        this.info=info;
    }
    public string Info
    {
        get {return info;}
    }
}
```

C# - Události (3)

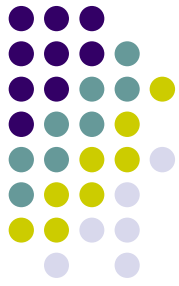


```
class Producer {
    string name;
    public Producer(string name) {
        this.name=name; }
    public string Name {
        get {return name;} }
    public delegate void WantToKnow(Producer source,InfoEventArgs
args);

    public event WantToKnow ItemProduced;

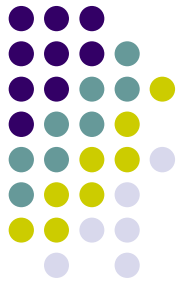
    public void Produce(string productName) {
        Console.WriteLine("Production of "+productName+" started.");
        InfoEventArgs info=new InfoEventArgs(productName);
        Console.WriteLine("Production of "+productName+" ended.");
        //vyvolání události
        if (ItemProduced!=null) ItemProduced(this,info);
    }
}
```

C# - Události (4)



```
class Customer
{
    string name;
    public Customer(string name, Producer producer) {
        this.name=name;
        //registrace
        producer.ItemProduced+=new
            Producer.WantToKnow(NewItemProduced) ;
    }
    public void NewItemProduced(Producer producer, InfoEventArgs
    info)
    {
        //skutečná obsluha události
        Console.WriteLine(this.name+": "+producer.Name+"
            produce item:"+info.Info) ;
    }
}
```

C# - Události (5)



```
class RunApp
{
    public static void Main()
    {
        Producer producer=new Producer("Haven inc.");

        Customer marek=new Customer("Marek",producer);
        Customer tom=new Customer("Tom",producer);

        producer.Produce("Ferrari");
        producer.Produce("pencil");
        producer.Produce("cake");
    }
}
```



C# - Události (6)

```
Production of Ferrari started.  
Production of Ferrari ended.  
Marek: Haven inc. produce item:Ferrari  
Tom: Haven inc. produce item:Ferrari  
Production of pencil started.  
Production of pencil ended.  
Marek: Haven inc. produce item:pencil  
Tom: Haven inc. produce item:pencil  
Production of cake started.  
Production of cake ended.  
Marek: Haven inc. produce item:cake  
Tom: Haven inc. produce item:cake
```



C# - Atributy

- Platforma .NET definuje možnost asociovat libovolné informace (metadata) se zdrojovým kódem aplikace - atributy.
- Tyto metadata jsou pak součástí assembly.
- Tyto metadata jsem v aplikaci schopni získat pomocí mechanismu reflexe.
- Příklad
 - [assembly: AssemblyVersion("1.0.*")]

Komponenty v .NET - Rozdíly mezi COM a .NET



- Neexistuje žádné základní rozhraní jako *IUnknown*.
- Komponenty se vytvářejí přímo, ne přes *class factory* (metoda *CoCreateInstance*).
- Nepoužívají se čítače odkazů.
- Všechny informace o rozhraní jsou obsaženy ve zdrojovém programu, nepoužívá se IDL.
- Identifikace komponent nevychází z GUID, ale z namespace a pro sdílené komponenty i z digitálního podpisu (strong name).

Komponenty v .NET - Distribuce komponent



- Základní jednotka je **assembly**
 - Jednodušší instalace pouhým kopírováním.
 - Jednotka verzování a zabezpečení
- Forma DLL nebo EXE
 - Předchází problémům souhrnně označovaným jako *DLL Hell*.
- Může obsahovat několik fyzických modulů
 - Komponenty – každá třída může být komponentou
 - Další zdroje – ikony, obrázky, řetězce
 - Metadata + manifest
- Assembly Linker – AL.exe

Komponenty v .NET - Manifest Assembly



Manifest je blok metadat obsahující informace

- Identitu – jméno, verze a kultura;
- Seznam souborů + kryptografické zabezpečení;
- Odkaz na další použité assembly + jejich verze;
- Exportované (veřejně viditelné) typy a zdroje;
- Bezpečnostní požadavky:
 - Nutné pro spuštění assembly;
 - Doporučené pro běh;
 - Ty, které by neměly být nikdy přiděleny.

Komponenty v .NET - Instalace .NET aplikací a komponent



- Soukromé assembly
 - Instalace typu XCopy.
 - Manifest obsahuje všechny potřebné údaje.
- Sdílené assembly
 - Nejčastěji instalovány do GAC – Global Assembly Cache.
 - Složitější instalace i odinstalování aplikace.
 - Definuje sdílené jméno – strong name.

Komponenty v .NET - Strong name (1)



- Pojmenování pro podepsanou assembly
- Významný bezpečnostní prvek
- Jen assembly s podepsaným jménem mohou být v GAC
- Politika verzí
- Utilita sn.exe

Komponenty v .NET - Strong name (2)



- Pomocí IDE

- [assembly: AssemblyKeyFile(<cesta k souboru s podpisem>)]
- [assembly: AssemblyKeyName(<název klíče v úložišti klíčů>)]
- [assembly: AssemblyDelaySign(<>true|false>)]

- Příkazovou řádkou

```
csc.exe /out:TestSignedAssembly.mod /t:module /r:TestSignedLibrary.dll
      TestSignedAssembly.cs AnotherClass.cs

al.exe /out:TestSignedAssembly.exe /t:exe /win32icon:App.ico
      /main:TestSignedAssembly.TestSignedAssembly.Main
      /keyf:LNavratCert.snk
      /link:obrazek.bmp TestSignedAssembly.mod
```

Komponenty v .NET - Strong name (3)



```
.module extern TestSignedAssembly.mod
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\U.4..
    .hash = (E6 8E F4 00 2B 3C 3C 88 D6 32 F2 72 A3 22 FA C8 // .....+<<...2.r."..
            A7 7B 24 07 ) // .{$.
    .ver 1:0:5000:0
}
.assembly extern TestSignedLibrary
{
    .publickeytoken = (45 A3 81 B3 3D EE F2 A7 ) // E...=...
    .hash = (CD 30 93 40 E6 D3 F9 69 95 E0 81 E0 62 07 07 0E // .0.@...i%...b...
            B6 09 3E 48 ) // ..>H
    .ver 1:0:1697:20437
}
```

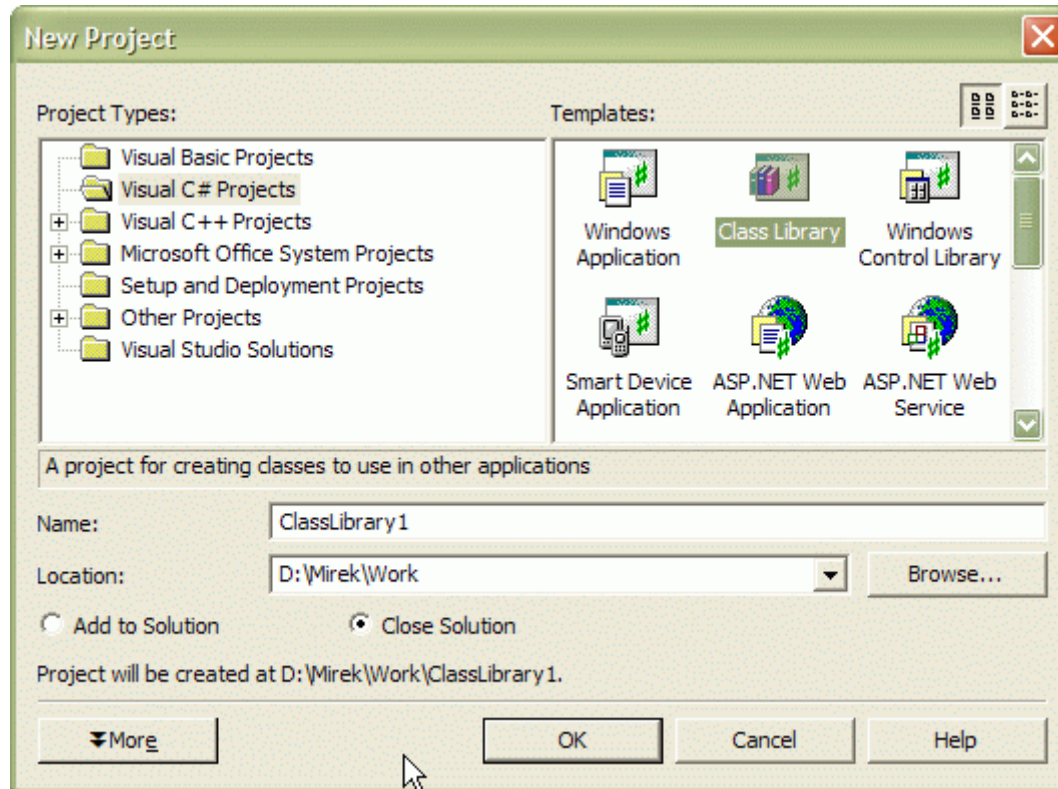
Posledních 8B hash hodnoty z veřejného klíče

Komponenty v .NET - Komponenty ve Visual Studio (1)

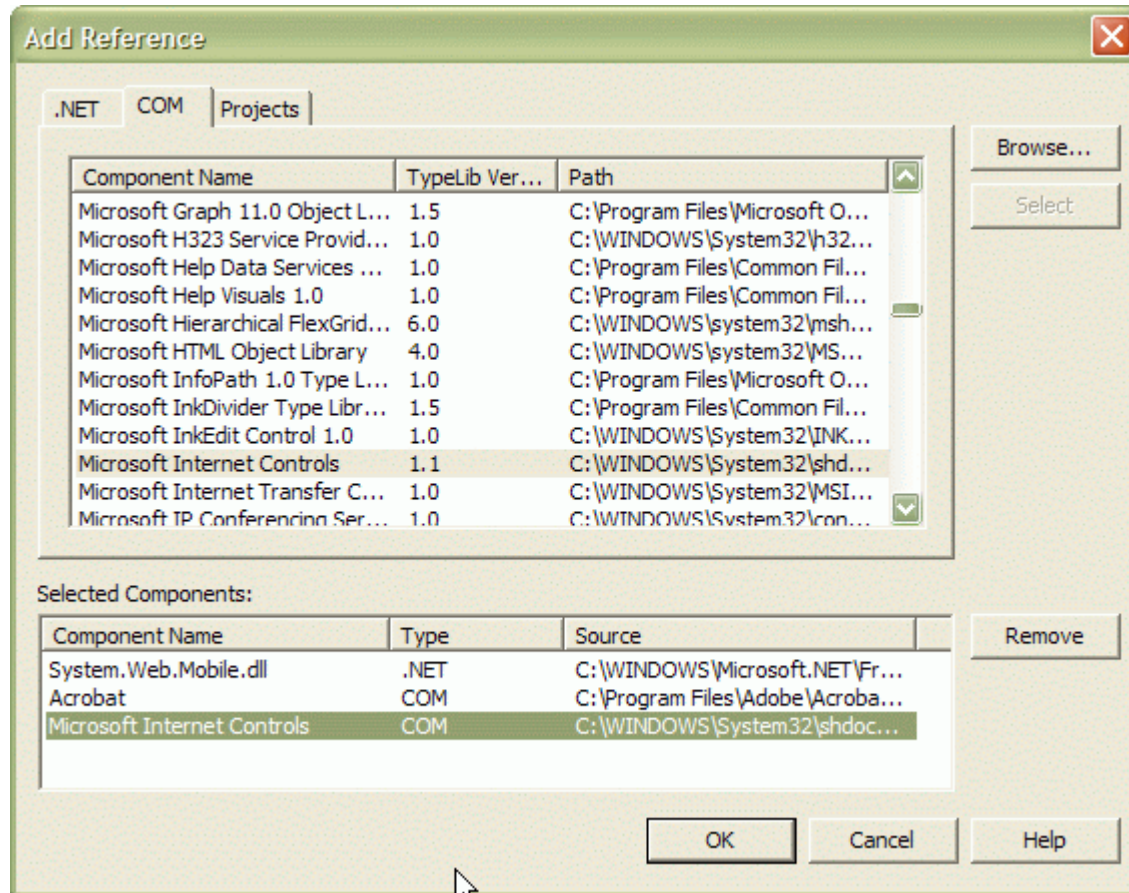
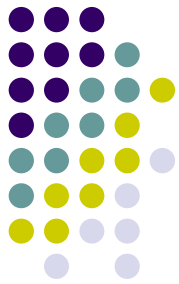


- **Vytváření komponenty**
 - Projekt typu Class Library
- **Použití komponenty**
 - Add Reference – import definice typů a komponent z jiných knihoven (systémových nebo vlastních)
- **Main program**
 - Windows Application – GUI
 - Console Application – příkazový řádek

Komponenty v .NET - Komponenty ve Visual Studio (2)



Komponenty v .NET - Komponenty ve Visual Studio (3)



Komponenty v .NET - Programování založené na rozhraních (1)



- **Rozhraní** = specifikace kontraktu
 - Referenční typ
 - Má pouze veřejné členy
 - metody, vlastnosti, indexery, události
 - Nemá implementaci, konstruktory, a nelze vytvářet instance
- **Class** = implementace kontraktu
 - Může dědit pouze z jedné třídy
 - Může implementovat více rozhraní
 - Může navíc obsahovat proměnné, konstanty, ...

Komponenty v .NET - Programování založené na rozhraních (2)



- Implicitní přetypování

```
public interface IMyInterface {  
    void Method1();  
}  
public class MyClass: IMyInterface {  
    void Method1() { ... }  
}
```

```
IMyInterface obj =  
    (IMyInterface) new MyClass();  
obj.Method1();
```

Komponenty v .NET – Příklad (1)



```
using System;
namespace PTE {

    public class Counter {

        // event - maximal value reached
        public event EventHandler LimitReached;

        public Counter(int limit) {
            this.limit = limit;
        }
    }
}
```

Komponenty v .NET – Příklad (2)



```
// counter value
private int val;
public int Value {
    get { return val; }
    set { val = value; }
}

// maximum value
private int limit;
public int Limit {
    get { return limit; }
    set { limit = value; }
}
```

Komponenty v .NET – Příklad

(2)



```
// increment counter and check overflow
public void increment() {
    val++;
    if( val >= limit ) {
        if( LimitReached != null )
            LimitReached(this, EventArgs.Empty);
        val = 0;
    }
}
}
```

Komponenty v .NET – Komponenty GUI v .NET (1)



- namespace **System.Windows.Forms**
 - komponenty pro lokální aplikace
 - nahrazuje C++ MFC (MS Foundation Classes)

- namespace **System.Web.UI**
 - komponenty pro webové aplikace
System.Web.UI.HtmlControls
System.Web.UI.WebControls
 - součást ASP.NET

PTE - Microsoft Visual C# .NET [design] - Form1.cs [Design]*

File Edit View Project Build Debug Data Format Tools Window Help

Debug FindControl

Server Explorer

App.cs | Counter.cs | **Form1.cs [Design]***

Form1

Login: textBox1

Password: textBox2

OK

Solution Explorer - PTE

Solution 'PTE' (1 project)

- PTE
 - References
 - App.cs
 - App.ico
 - AssemblyInfo.cs
 - Counter.cs
 - Form1.cs

Solution... Class... Index

Properties

button1 System.Windows.Forms.Button

ImageAlign	MiddleCenter
ImageIndex	(none)
ImageList	(none)
RightToLeft	No
Text	OK
TextAlign	MiddleCenter

Behavior

Text

The text contained in the control.

Properties Dynamic Help

Output

Debug

```
'DefaultDomain': Loaded 'c:\windows\microsoft.net\framework\v1.1.4322\mscorlib.dll', No s
'PTE': Loaded 'D:\Mirek\Work\PTE\bin\Debug\PTE.exe', Symbols loaded.
The program '[2924] PTE.exe' has exited with code 0 (0x0).
```

Task List Output Index Results

Ready

Komponenty v .NET – Komponenty GUI v .NET (3)



```
Windows.Forms.Label label1 =  
    new Windows.Forms.Label();  
  
label1.Location = new Drawing.Point(24, 24);  
label1.Name = "label1";  
label1.Size = new Drawing.Size(56, 24);  
label1.TabIndex = 0;  
label1.Text = "Login";  
  
Controls.add(label1);
```

Komponenty v .NET – Spolupráce s existujícím kódem



- Autoři platformy .NET a jazyka C# umožnili programátorům použít stávající programy. Používáme-li takovéto programy, zřikáme se výhod, které poskytuje CLR. Hovoříme pak o neřízeném kódu.
 - Spolupráce s komponentami modelu COM - schopnost prostředí .NET používat komponenty modelu COM a naopak komponentám modelu COM používat prvky prostředí .NET.
 - Spolupráce s běžnými knihovny DLL - tyto služby umožňují programátorům v prostředí .NET používat knihovny DLL.
 - Nezabezpečený kód - umožňuje programátorům v jazyce C# používat například ukazatele. Takto vytvořený program není spravován CLR systému .NET.

Komponenty v .NET – COM interoperability (1)



Použití existující COM komponenty v .NET aplikaci.

- Nejprve je nutné COM komponentu registrovat v registry (nástroj regsvr32.exe).
- Vytvoříme nový projekt v jazyce C#. V okně Solution Explorer aplikace Visual Studio vybereme položku Add Reference.
- Najdeme příslušnou COM komponentu.
- Reference na tuto komponentu byla přidána a lze ji normálně používat (tedy instanciovat a pak volat její metody,.....).

Komponenty v .NET – COM interoperability (2)



- Řekněme, že máme zaregistrovanou COM komponentu *example.dll*. Tato komponenta je přístupná přes rozhraní *IExample*.
- interface

```
Interface IExample {  
    int getNumber();  
}
```

Reference na tuto komponentu pak bude:
EXAMPLELib.Example.

```
EXAMPLELib.Example  
example = new EXAMPLELib.Example();
```

Komponenty v .NET – COM interoperability (3)



Použití .NET komponenty na místo COM komponenty.

- Vytvoříme nový C# projekt typu Class Library.
- Komponentu musíme zaregistrovat v Registry. K tomu složí nástroj regasm.exe.
- Pro programy, které pracují s typovanými knihovnamí lze vygenerovat potřebný .tlb soubor nástrojem tlbexp.exe
- Pak lze tuto komponentu využívat stejně jako COM komponentu.

Komponenty v .NET – Spolupráce s běžnými knihovnamí DLL (1)



- Platform Invocation Services (PInvoke) - tyto služby umožňují řízenému kódu pracovat s knihovnamí a funkcemi exportovanými z dynamických knihoven.
- Knihovna DLL je importovaná pomocí atributu DllImport.
- Importované funkce musí být označeny jako externí (klíčové slovo extern).

Komponenty v .NET – Spolupráce s běžnými knihovnamí DLL (2)



```
using System;
using System.Runtime.InteropServices;
class RunApp
{
    [DllImport("user32.dll")]
    static extern int MessageBoxA(int hWnd, string msg, string
caption, int type);

    static void Main(string[] args)
    {
        MessageBoxA(0, "Hello world!", "C# is calling!", 0);
    }
}
```