Sémantická mezera

**klasické programování**(procedurální programování)

-sekvence po sobě prováděných příkazů

**objektově orientované programování** (OOP)

-data uschována, metody, spolupráce objektů

# Základní termíny OOP

### **objekt**

„entita“ obsahující data a chování

**identita** - odlišuje objekt od jiných objektů

**stav** - popisuje datovou složku objektu

**chování** - popisuje metody spojené s protokolem objektu

protokol = množina všech zpráv, které lze objektu zaslat a ten je na ně schopen reagovat

### třída

„šablona“ pro vytváření objektů

definuje složky stavu a chování objektu

instanční proměnné uchovávají data(stav)

metody uchovávají chování

**instance -** objekt vytvořený podle dané šablony

### zapouzdření

všechna data objektu jsou navenek neviditelná

data zpřístupněna pomocí „přístupových metod“ - getter/setter

**statická vs. dynamická složka**

### metoda

část chování objektu

**viditelná vs. neviditelná**

může realizovat dynamickou datovou složku

může zpřístupňovat statickou datovou složku (getter/setter)

zvnějšku spustitelná pouze pokud je svázána se zprávou, která je součástí protokolu

### abstrakce

zobecnění

obecné (abstraktní) třídy, obecná rozhraní

využití: tvorba složitějších třídních hierarchií, definice společného chování pro různé třídy

### dědičnost

-objekt může být založen na existujícím objektu

-umožňuje členění a znovupoužití kódu

**klasické dědění**

objekt definován třídou, třída dědí z jiné třídy

**dědění mezi objekty**

objekt dědí z jiného objektu

**nadtřída -** třída, ze které se dědí

**podtřída -** třída, která dědí

### polymorfismus

schopnost více objektů reagovat na stejnou zprávu

**klasický**

objektům A a B zašle objekt X zprávu m a oba na ní reagují

**stavový** (různé stavy téhož objektu)

objektu A zašle objekt X zprávu m v různém čase a objekt A reaguje různě podle svého současného stavu

**subjektivní**

objektu X zašlou zprávu m objekty A a B a X reaguje podle odesílatele

# Jazyk C#

Velikost písmen - C# je citlivý na velikost písmen, včetně názvu promměných a metod

## Pojmenovávání

* namespace - MujHttpNamespace
* assembly/sestavy - MojeVlastniAssembly
* třídy - MojeXmlTřída
* vyjímkové třídy - MojeVlastniException
* rozhraní - IMojeRozhrani
* properties/vlastnosti - MojeVlastnost
* parametry a proměnné - MojeProměnná
* private/protected proměnné - \_mojeProměnná

## Výrazy / statements

základní jednotka kódu

oddělení středníkem

-definice proměnné, výrazu, volání metody, řízení toku programu, vytvoření objektu, přiřazení hodnoty do proměnné

bloky výrazů -lze je vnořovat, tvoří hranice proměnných v nich deklarovaných

## Komentáře

umožňují vnitřní dokumentaci zdrojového kódu

kompilátor je ignoruje

**jednořádkové komentáře** uvozeny //

**víceřádkové komentáře** začínají /\* a končí \*/

všechen text mezi začátkem a koncem je komentářem

**komentáře pro XML dokumentaci**

využívány pro generování XML dokumentace

jednořádkové uvozeny ///

víceřádkové začínají /\*\* a končí \*/

## Proměnné

slouží k uchovávání hodnot

každá proměnná má explicitní typ

třídní/instanční proměnné - public, protected, internal, protected internal, private

lokální proměnné

parametry

## Typy

Char,int,byte,shortlong

Double,decimal,float

Bool,object,string

## Řídící struktury

řídí tok programu

podmíněné výrazy - if, switch

iterace - do, while, for, foreach

skoky - break, goto, return, yeild

**IF -** podmíněný výraz

if („podmínka“) if-tělo [else else-tělo]

**SWITCH -** větvení podle hodnoty řídící proměnné

**DO -** do („tělo“) while („podmínka“)

**WHILE -** while („podmínka“) tělo

**FOR -** for („inicializace“, „podmínka“, „iterace“) tělo

**FOREACH -** iterace v enumerovatelné sekvenci

## Namespace

utváří „kontext“ pro pojmenovávání

umožňují vytvářet třídy stejných názvů jako jsou obsaženy v .NET frameworku

## Třídy (class)

šablony pro tvorbu objektů

obsahují implementaci funkcionality programu

vytváření instancí

vlastnosti:

zprostředkovávají funkcionalitu metod pomocí syntaxe pro proměnné

statické vs. instanční

klasická vs. zjednodušená syntaxe pro zpřístupňovací metody

### kontruktor

slouží k řízení inicializace objektu

většinou nastavují výchozí stav vlastností

lze využít k naplnění konkrétních hodnot

### úrovně přístupnosti

**private**

přístupné pouze v rámci třídy

lze je měnit jen s pomocí metod a konstruktoru dané třídy

**protected**

přístupné v rámci třídy a jich potomků

**public**

přístupné pro všechny metody ve všech třídách

**internal**

přístupné pouze v rámci assembly

### Statické třídy

využívají se mj. k implementaci vzoru „jedináček“

všechny metody, vlastnosti a proměnné jsou také statické

není nutné vytvářet instanci

### Parciální třídy

umožňují rozdělit definici třídy do více souborů

### Abstraktní třída

něco mezi rozhraním a klasickou třídou

definuje stav a a chování - na rozdíl od rozhraní však může pro libovolnou část definovaných prvků obsahovat i implementaci

## Dědění

schopnost vytvořit třídu z jiné třídy a rozšířit chování nebo stavovou složku této třídy

možnost přetížení metod předka

důležité vlastnosti dědění:

* + - * potomek rozšiřuje předka ¨C obsahuje vše co předek a něco navíc
      * odvozená třída nemůže měnit definici poděděného prvku
      * dědí se vše kromě konstruktorů a destruktorů
      * prvky odvozené třídy mohou být předefinovány (override)

class odvozena: zakladni

new/virtual

override

## Rozhraní - Interface

množina definicí metod a vlastností

název začíná I

implementace rozhraní ¨C děděním a implementací jeho metod a vlastností

rozhraní může dědit z rozhraní, ale ne z třídy

třída využívající rozhraní musí implementovat vše co je jím definováno

## Delegáti a události

základní kameny událostmi řízeného programování

**delegát** - definuje, která metoda je volána pokud je odpálena událost

**událost** - oznámení, že byla provedena nějaká akce

### Delegáti

datový typ umožňující referencovat metodu

umožňují multicasting

definice delegáta určuje signaturu přiřazené metody

## Výjimky a jejich ošetřování

umožňují ošetřování chyb

oddělení normálního kódu od ošetřování chyb

výjimka (exception) představuje takřka jakoukoliv anomální situaci, která může nastat

kód detekující chybu „hází“ výjimku (throw), kód ošetřující nastalý stav výjimku „chytá“ (catch)

všechny výjimky instancí System.exception nebo jejích potomků

výjimky definovány v .NF, lze vytvářet vlastní

**try/catch -** zkus něco udělat a když nastane chyba, tak ji chytni

**try/catch/finally -** zkus něco udělat a když nastane chyba, tak ji chytni a VŽDY udělej to, co ej ve finally

**try/finally -** zkus něco udělat, ale VŽDY udělej to, co je ve finally ¨C vyjímky se vyhodí až finally proběhne

výjimky se chytají zdola nahoru - FileNotFoundException > IOException > SystemException > Exception

catch jde řetězit

## Enumerace

výčtový datový typ

každá položka má jméno a pod sebou hodnotu podle které se řadí (většinou int)

## Pole

množina objektů stejného typu

délka je dána hodnotou nebo proměnnou při deklaraci

po deklaraci nelze délku měnit

## Generické typy

umožňují využít jednu definici pro několik datových typů

maximalizují znovupoužitelnost kódů, typovou bezpečnost a výkon

definice může obsahovat typové omezení - klíčové slovo where

### Kolekce

slouží k seskupení dat

klasické i generické třídy kolekcí

většina má stejné rozhraní

**pokud potřebujeme FIFO** (first-in, first-out) – Queve

**pokud potřebujeme LIFO** (last-in, first-out) - Stack

**přístup indexem** - ArrayList, StringCollection, List

**přístup pomocí klíče** - Hashtable, SortedList, Dictionary,…

## Data, LINQ

práce s daty na databázovém serveru

ADO.NET

MS SQL Server

OLE DB

ODBC

Oracle

data přímo součástí aplikace (kolekce)

### Struct

složitější datová struktura (odelhčený objekt)

* při předání jako parametr se předávají hodnotou
* nepodporují dědění
* vždy musí mít implicitní kontruktor
* efektivní pouze pro menší objemy dat

### LINQ

Dotazovací jazyk uvnitř kódu

Language INtegrated Query

**LINQ to Objects**(Manipulace s kolekcemi a poli objektů

Syntaxe velmi podobná SQL), LINQ to SQL, LINQ to DataSet, LINQ to XML

Pracuje s poli a kolekcemi implementujícími IEnumerable<T>, s XML a SQL databázemi

## Grafika

### GDI+

Základní mechanismus Windows

Nativní podpora přímo v .NET

Neakcelerovaná grafika

**Funkce:**

* System.Drawing - Základní funkcionalita
* System.Drawing.Drawing2D - Funkce pro podporu 2D a vektorové grafiky
* System.Drawing.Imaging - Podpora pro práci s obrázky
* System.Drawing.Text - Typografické funkce
* System.Drawing.Graphics - Představuje kreslící plochu

Obsažen v každém vizuálním prvku

Přístup pouze:

Přes vlastnost PaintEventArgs.Graphics u události Paint

Zavoláním metody CreateGraphics() u formuláře nebo ovládacího prvku

Pomocí metody Graphocs.FromImage() z existujícího obrázku

**Pen**

**Brush**

Definuje výplň

#### Perzistence

Opakované vykleslení v metodě Paint()

Může být náročné -> Blikání

**Bitmapa v paměti -** Vykreslení na bitmapu v paměti (není potřeba opakovat)**;** V Paint() vykreslíme jen bitmapu

**Double buffering -** Kreslíme na „plátno“ v paměti a teprve po vykreslení zobrazíme

### WPF

Akcelerovaná grafika

### DirectX/ Direct3D

### Direct2D

Akcelerovaná 2D grafika – Windows Vista + 7

Není zatím dostupné v C#

### XNA

Nativní podpora C#

XNA Game Studio

## WPF (Windows Presentation Foundation)

UI framework

Kombinuje:

* Aplikační rozhraní
* 2D grafiku
* 3D grafiku
* Využití dokumentů
* Multimědia

Oddělení definice UI od logiky

**Definice UI** – XAML (Extensible Application Mark-up Language)

**Logika** – C#, Visual Basic

Vektorové vykreslování

Výhody:

* Volná vazba mezi vzhledem a chováním
* Designéři UI a vývojáři mohou pracovat s oddělenými modely
* Grafické vývojové nástroje nemusí pracovat na úrovni zdrojového kódu ale s čitelným XML dokumentem

### XAML

Jazyk založený na XML

Umožňuje inicializaci objektů v hierarchické struktuře

XAML se snadněji čte a definice v něm je kratší nez v kodu

Oddělení designu a logiky

Vlastnost (property) jako element

Implicitní typová konverze

# Návrh rozložení

Vyhněte se definici pevných pozic

Vyhněte se pevným velikostem

Využívejte správně jednotlivé panely (Stack panel, Grid Panel,..)

Využívejte Margin (místo okolo prvku) a Padding (místo uvnitř prvku)

Ořezávání (clipping) - Panely implicitně ořezávají obsah, lze vypnout;Vlastnost ClipToBounds

# Návrhový vzor

obecné řešení nějakého problému při návrhu programu

popis řešení nebo obecná šablona

nejedná se o knihovnu nebo kus kódu¨

**tvořivé** (creational)

řeší problémy související s vytvářením objektů v systému

**strukturální** (structural)

zaměřené na uspořádání tříd nebo komponent systému

cílem je zpřehlednit a strukturovat kód

**behaviorální** (behavioral)

týkají se chování systému

## Singleton

Jedináček

nejjednodušší návrhový vzor

jeho cílem je zajistit, že existuje pouze jedna instance dané třídy a poskytnout metodu, která ji zpřístupňuje

* Instancuje se pouze jednou
* Má private nebo protectedKonstruktor bez parametrů
* jediný přístup k instanci přes statickou metodu
* nelze z něj dědit

## AbstractFactory

tvořivý vzor

zapouzdřuje množinu podobných „továren“ vytvářejících rodinu objektů

při jeho použití není nutné specifikovat danou třídu

## Adapter

Adaptér

strukturální vzor

mění rozhraní třídy na rozhraní očekávané klientem

## State

Stav

behaviorální vzor

změna chování objektu při změně jeho vnitřního stavu

využívají se třídy zapouzdřující rozdílné chování v závislosti na stavu objektu

objekt zná svůj aktuální stav a využívá jeho metody

neřeší, kdo je zodpovědný za určení aktuálního stavu

# Refactoring

techniky úpravy existujícího kódu za účelem změny

jeho struktury bez změny funkčnosti

refaktorováním můžeme získat: větší udržovatelnost - kód je snáze čitelný, lze v něm lépe a rychleji hledat chyby; větší rozšiřitelnost

Motivace – pachy v kódu

## pachy v kódu

symptomy problému ve zdrojovém kódu

duplikovaný kód, dlouhá metoda, velká třída, líná třída, příliš mnoho parametrů

## Refactoring

techniky pro zvětšení abstrakce:

* zapouzdření proměnné - proměnná je zpřístupněna jen pomocí getteru a setteru
* náhrada typové kontroly stavem nebo strategií
* náhrada podmíněného výrazu polymorfismem

techniky pro dělení kódu:

* vyčlenění metody
* vyčlenění třídy

techniky ovlivňující podobu kódu:

* přesun metody nebo proměnné
* přejmenování metody nebo proměnné
* přesun metody/proměnné do nadtřídy
* přesun metody/proměnné do podtřídy

# Unit Testing

testování kódu po částech

cílem je ověřit, že kód dělá co by měl