Namespace

- Namespace je jmenný prostor,
- syntakticky vypadá podobně jako třída,
- určuje území platnosti identifikátorů,
- lze vnořovat namespace do namespacu,
- nelze vnořit namespace do třídy.

Namespace II

- U nelokálních identifikátorů je nutno říct, ve kterém namespacu jsou,
- chceme-li používat identifikátory z konkrétního namespace častěji, můžeme použít `using`, například: `using System;`
- Příklad správně: `using System;`
  `Console.WriteLine();`
- Příklad správně: `System.Console.WriteLine();`
- Příklad špatně: `using System.Console;`
  `WriteLine();`
  špatně, protože `Console` je statická třída, ne jmenný prostor.

Namespace III

- Pomocí `using` lze vytvářet aliasy tříd:
- `using <alias>=<třída>;`
- Příklad: `using c = System.Console;`
  `c.WriteLine();`
- Nyní už rozumíme celému obrázku, který Visual Studio vygeneruje,
- až na to, že ne nutně víme, co nám jednotlivé namespacy umožňují,
- zájemci to ovšem mohou různými způsoby zjišťovat.

Vstup a výstup

Textový soubor

- Standardní vstup se řídí ze třídy `Console` v namespacu `System`.
- Textové soubory se ovládají velmi podobně pomocí StreamReaderu a StreamWriteru.
- `StreamReader a StreamWriter` jsou v namespacu `System.IO`, tedy:
- `System.IO.StreamReader r= new System.IO.StreamReader(@"c:\temp\file.txt");`
- Instance třídy `StreamReader` resp. `StreamWriter` má stejný charakter jako v Pascalu proměnná typu `text` assignovaná ke konkrétnímu souboru.
- V Pascalu jsme jméno proměnné typu `text` předávali každé funkci, v C# voláme dotyčnému readeru (resp. writeru) metody

Vstup a výstup II

Textový soubor některé metody tříd `StreamReader a StreamWriter`

- `void r.Close() // zavře reader,`
- `string r.ReadToEnd() //přečte soubor do konce,`
- `w.Write(co) //vypíše do writeru,`
- `r.Read() // přečte znak z readeru,`
- `r.EndOfStream // atribut určující konec vstupního streamu`
- @"....." // pevný řetězec – nefungují v něm sekvence $\backslash n, \backslash r, ...$

Vstup a výstup II

Příklad

Zkopírování souboru

```
System.IO.StreamReader r=new
System.IO.StreamReader("soubor.txt");
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt");
w.Write(r.ReadToEnd()); w.Close();
```

Vstup a výstup III

Příklad

Zkopírování souboru po znacích `System.IO.StreamReader`

```
r=new System.IO.StreamReader("soubor.txt");
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt");
while(!r.EndOfStream) w.Write((char)r.Read());
w.Close();
```

Poznámky

- Metody `ReadLine()`, `resp. WriteLine()`,
- kódování češtiny – lze předat jako druhý parametr konstruktoru Readeru/Writeru:
- Příklad:

```
Encoding e=Encoding.GetEncoding(1250);
Encoding f=Encoding.GetEncoding(852);
System.IO.StreamReader r=new
System.IO.StreamReader("soubor.txt",e);
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt",f);
w.Write(r.ReadToEnd()); w.Close();
```

Počítačová simulace

- Máme úlohu dostatečně těžkou k představení, chceme si vytvořit názor.
- Simulovat lze různé věci (úraz – tedy třebas jeho hojení, šíření alkoholu v organismu, jízdu výtahů s lidmi...).
- Počítačová simulace je simulace, při níž modelem je (počítačový) program.
- Úkolem programu je zjistit, jak se bude simulovaný systém chovat.
- Úkolem není situaci optimalizovat!
- Výsledky simulace mohou být různé, základním výsledkem je čas konce simulace.

Počítačová simulace II

- Simulace spojitá VS diskrétní,
- při spojité simulaci je zpravidla potřeba vyřešit soustavu rovnic (nezřídka diferenciálních),
- nás bude zajímat simulace diskrétní,
- spojitou simulaci s jejím použitím lze zkusit aproximovat malými kroky a častým přepočítáváním (k tomu je ovšem potřeba stabilita řešení).

Auta s pískem

Velmi typická úloha na přednáškách programování

- Chceme převézt hromadu písku na staveniště,
- máme k dispozici jistý počet dělníků a jistý počet aut,
- na cestě mohou být kritické sekce (u hromady a na stavbě také),
- dělníků je omezený počet, na vybraných úsecích může být zákaz předjíždění nebo dokonce jeden jízdní pruh pro oba směry.
- Jak rozdělit dělníky a naplánovat auta tak, aby bylo převezeno co nejdříve?
- Diskrétní simulace bude simulovat konkrétní rozvrh (dělníků a aut) a tedy určí, za jak dlouho bude hromada odvezena.
- Obvykle uvažujeme jedno místo, kde je provoz sveden do jednoho jízdního pruhu pro oba směry.

Výtahy

- Podle D. Marxe (bývalého vedoucího MERL) je potřeba, aby výtahy přijely do půl minuty od zavolání,
- testy na živých lidech lezou výrobci do peněz (zákazníci utíkají),
- proto se hodí prostředí simulující požadavky lidí v domě na výtah.
- Zajímavá je distribuce dob čekání jednotlivých lidí.

Samoobsluha

- Jak v samoobsluze rozmístit zboží, aby lidé museli při běžném nákupu prolézt celý podnik (s vyhlídkou na to, že třebas koupí neco, co původně nechtěli)?
- Kolik lidí naštvou příliš dlouhé fronty u pokladen a kolik lidí tudíž raději nakoupí jinde (a po kolika bude nutné uklízet nákupní košíky plné zboží)?
- Nakupující přecházejí po samoobsluze podle toho, co chtějí koupit,
- s množstvím vybraného zboží zpravidla roste trpělivost zákazníka.

Obecné řešení diskrétní simulace

V objektovém prostředí

- Všimneme si, že není třeba se starat o dobu, kdy nějaký proces běží,
- zajímavé je jen kdy proces začne/skončí. Proto sledujeme pouze tzv. události.
- Obvykle pro jednotlivé účastníky naprogramujeme třídy, které je nějakým způsobem reprezentují (pomocí atributů a metod),
- vyrobíme kalendář událostí (pomocí kterého se orientujeme v dění),
- vyrobíme simulační jádro schopné obsloužit konkrétní událost.

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přeplánovávat).
- Simulační jádro z kalendáře vytáhne první událost (tedy tu, která nastane nejdříve ze všech),
- v rámci obsluhy události můžeme přidávat další události nebo rušit naplánované události.
- Kalendář je též typicky schopen měřit čas (určit čas simulace).
- Simulace končí, pokud po obsluze události zůstane kalendář prázdný.

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:
- Běží (aktivní) – právě se obsluhuje jeho událost,
- naplánovaný – čeká v kalendáři událostí,
- čeká (pasivní) – čeká, až ho někdo vzbudí,
- ukončený – doběhl a nebude mít další události.

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo. Jak to udělat?
- Buďto čekající auto přejde do stavu `čeká`, nebo si spočítá čas, kdy projíždějící auto odjede a na tu dobu se naplánuje.
- V prvním případě musí auto někdo probudit,
- ve druhém případě musí znovu zjistit, zda je kritická sekce volná.
- Race condition – aneb proč (a kdy) mohou auta nabourat?

Způsoby řešení situací II

- Co když program běží ve více procesech a mezi okamžiky, kdy jedno auto zjistí, že KS je volná a vjede do ní, se na totéž zeptá další auto.
- Toto je obecný informatický problém zvaný "race-condition".
- Co když auto zjistí, že kritická sekce je obsazena, ale než se zafrontuje, projíždějící auto ji opustí?
- Pak jde o jistou variantu problému uváznutí.