Náhodnost

Náhodné je v našem vesmíru pouze málo věcí. Obecně musíme často pravou náhodnost skrývat za nepravdivou a to tím, že náhodné čísla generujeme z různých parametrů. Existují totiž různé funkce či řady, které umí dle zadaných parametrů vždy vygenerovat stejné čísla za sebou, které vypadají náhodně. Jedná se např. o Blum Blum Shub či Lineární kongruentní generátor. Tyto generátory generují pseudonáhodné čísla. To jsou čísla, která jsou závislé na svém pořadí volání a svých vstupních parametrech.

Vstupní parametry mohou být různých zdrojů. Často se používá čas (na milisekundy či nižší) a třeba i pohyb myše po obrazovce. Obecně těmto parametrům říkáme seed. Ten je třeba i ve hře Minecraft, kterou dobře znáte. Z této hry taktéž víte, že když zadáte stejný seed, tak se vám i na jiném PC v jiný čas vygeneruje stejný svět. To je nevýhody pseudonáhodných generátorů, protože když víte počet volání čísel a jejich vstupní seed, tak umíte najít následující číslo.

V C# máme pro náhodné čísla vytvořený objekt / třídu Random. Tuto třídu si musíme připravit a poté z ní můžeme získávat náhodné celé i desetinné čísla.

```
Random random = new Random();
```

Na tomto objektu poté máme k dispozici řadu metod:

- `Next()` - vrátí náhodné číslo v rozsahu datového typu int.
- `Next(max)` - vrátí náhodné číslo v rozsahu 0 až max.
- `Next(min, max)` - vrátí náhodné číslo v rozsahu min až max.
- `NextDouble()` - vrátí desetinné číslo datového typu double v rozsahu 0.0 až 1.

Otázkou často bývá, jak získáme např. desetinné číslo v rozsahu 0 až 200. Jde to jednoduše, můžeme totiž výsledek `NextDouble` právě číslem vynásobit a to nám roztáhne maximální hodnotu:

```
Random random = new Random();

double output = random.NextDouble() * 200;

Console.WriteLine(output); // 0.0 - 200.0
```

Často se studentům stává, že vytvoří např. v cyklu vícekrát za sebou náhodný generátor číslem pomocí randomu. My však víme, že jako vstupní parametr si tato třída (bez našeho vědomí) bere právě datum a čas počítače. Vytvořením více instancí Random mám tedy bude generovat stejná čísla. Obecně se doporučuje na jeden program mít právě jeden náhodný generator.

```
Random random1 = new Random();
Random random2 = new Random();


Console.WriteLine(random1.Next(1, 10));
Console.WriteLine(random2.Next(1, 10)); // stejné číslo jako ve výpisu výše
```

Soubory a složky

Soubory snad každý zná. Jedná se o způsob uložení dat na počítači. V programech často potřebujeme z programů zapisovat a nebo číst tyto soubory. V C# máme dva způsoby k zapisování souborů. Oba si ukážeme. První je jednodušší.

V C# máme předpřipravenou třídu `File`, na které můžeme volat různé metody. Pro použití musíme importovat knihovnu `System.IO`. Metody k dispozici:

- `File.ReadAllText(cesta)` - vrací string, který obsahuje všechno z daného souboru
- `File.ReadAllLines(cesta)` - vrací pole stringů, který obsahuje jednotlivé řádky souboru
- `File.Exists(cesta)` - vrací bool, který zkontroluje, že daný soubor existuje
- `File.Delete(cesta)` - smaže soubor, pokud existuje, pokud ne, nic se nestane
- `File.WriteAllText(cesta, obsah)` - zapíše do daného souboru celý string (přepíše soubor)
- `File.WriteAllLines(cesta, obsah)` - zapíše do souboru dané řádky, typu stringového pole (přepíše soubor)
- `File.AppendAllText(cesta, obsah)` - přidá obsah (string) do daného souboru
- `File.AppendAllLines(cesta, obsah)` - přidá obsah (stringové pole) do daného souboru

Bez dalšího popisu máme k dispozici následující metody:

- `File.Copy(src, dst)` - zkopíruje soubor do dané složky
- `File.Move(src, dst)` - přesune daný soubor
- `File.Replace(src, dst, dstBackup)` - nahradí soubor na dané pozici, a jestli existuje, tak se daný přesune do zálohy
- `File.GetLastAccessTime(file)` - poslední čas přístupu
- `File.GetLastWriteTime(file)` - poslední čas zapsání

Když v nějaké metodě čteme do souboru, který neexistuje, vyhodí se výjimka `FileNotFoundException`.

Soubory již upravovat umíme, máme i způsob jak pracovat se složkami. Přístup je velmi podobný, jen máme k dispozici třídu `Directory`. Máme k dispozici metody:

- `Directory.CreateDirectory(path)` - vytvoří na daném místě složku
- `Directory.Exists(path)` - zkontroluje, zda složka existuje

- `Directory.Delete(path)` - smaže složku
- `Directory.Move(src, dest)` - přesune složku
- `Directory.GetFiles(path)` - získá soubory z dané složky
- `Directory.GetDirectories(path)` - získá složky z dané složky

Streamy

Čtení a zápis souborů pomocí třídy File má mnoho nevýhod. Jedna zásadní nevýhoda je ta, že často nevíme, jak bude soubor velký a tudíž nevíme, jestli se třeba vůbec vejde do paměti. Často se nám tedy hodí číst po částech. K tomu slouží proudy - streamy, které nám právě čtení či zapisování po částech dovolují.

Máme dva typy proudů - čtecí a zapisovací. Proudy po zapnutí způsobí to, že se soubor zablokuje a nikdo do něj v dané době nesmí zapisovat.

Prvně si ukážeme čtecí proud. Pro založení proudu musíme vytvořit instanci s jménem souboru:

```
StreamReader sr = new StreamReader("file.txt"); // vytvoří stream
string line;

while(true) // nekonečný cyklus
{
    line = sr.ReadLine(); // přečte další řádek a uloží do proměnné

    if(line == null) break; // pokud řádek neexistuje (jsme na konci souboru), cyklus skonč

    Console.WriteLine(line); // vypíše řádek do konzole
}

sr.Close(); // uzavře proud
// sr již nelze používat
```

Zavírání proudů je velmi nepřehledné a je velmi pravděpodobné, že zapomenete na jeho uzavření. Proto máme k dispozici blok `using`, který se o automatické uzavření postará automaticky. V případě, že bychom i bez něj proud neuzavřeli, tak se uzavře až se skončením procesu.

```
using(StreamReader sr = new StreamReader("file.txt"))
{
    string line;

    while((line = sr.ReadLine()) != null) // zmenšený zápis z toho předcházející ukázky
```

```
    {
        Console.WriteLine(line);
    }
}
```

Na proudu máme k dispozici ještě další způsoby čtení:

- `stream.Read()` - přečte následující část
- `stream.ReadLine()` - přečte vše do konce řádku
- `stream.ReadToEnd()` - přečte vše do konce souboru
- `stream.ReadBlock(buffer, index, count)` - přečte do bloku na daném indexu část

Zapisovací proud je podobný tomu ke čtení:

```
using(StreamWriter sw = new StreamWriter("cats.txt"))
{
    for (int i = 0; i < 400000; i++)
    {
        sw.WriteLine("yoko a bambi jsou kocouri");
    }
}
```

Datum a čas

Datum a čas v počítačích ukládáme v různých formátech. Nejčastěji se ukládá např. počet milisekund od nějakého data (UNIX, např. `63776293428232`) a nebo třeba formátově pomocí ISO stringů (např. `20011221T08:30:52Z`).

Pro prací s datem a časem máme v C# připravenou třídu `DateTime`. Vytvoření datumu:

```
// Format: rok, měsíc, den, hodina, minuta, sekunda, milisekunda
DateTime date = new DateTime(2008, 10, 1, 8, 30, 52);

DateTime basic = new DateTime();
```

Aktuální čas vytvoříme pomocí vlastnosti Now. Na instanci datumu máme taktéž k dispozici informace o daném datumu a času:

```
DateTime current = DateTime.Now;

Console.WriteLine(current.Day);
```

```
Console.WriteLine(current.Month);
Console.WriteLine(current.Year);
Console.WriteLine(current.Hour);
Console.WriteLine(current.Minute);
Console.WriteLine(current.Second);
Console.WriteLine(current.Millisecond);
Console.WriteLine(current.Ticks);

Console.WriteLine(current.Date); // vrací datum bez času
Console.WriteLine(current.DayOfWeek); // vrací den v týdnu, tzv. enum, např. DayOfWeek.Mond
Console.WriteLine(current.DayOfYear); // vrací den v roce
```

Datum můžeme upravovat pomocí pomocných metod. Pomocné metody nám vrací upravené datum toho, na kterém jsme metody zavolali.

```
DateTime modified1 = current.AddDays(3);
DateTime modified2 = current.AddMonths(1);
DateTime modified3 = current.AddYears(10);

DateTime modified4 = current.AddTicks(1000000);

DateTime modified5 = current.AddMilliseconds(10);
DateTime modified6 = current.AddSeconds(10);
DateTime modified7 = current.AddMinutes(10);
DateTime modified8 = current.AddHours(10);
```

Datum a čas můžeme formátovat:

```
Console.WriteLine(current.ToShortDateString());
Console.WriteLine(current.ToShortTimeString());
Console.WriteLine(current.ToLongDateString());
Console.WriteLine(current.ToLongTimeString());
Console.WriteLine(current.ToString());

// Pokročilé formátování, viz. https://docs.microsoft.com/cs-cz/dotnet/standard/base-types/
Console.WriteLine(current.ToString("dd. MM. yyyy"));
Console.WriteLine(current.ToString("d. M. yy"));
```

```
Console.WriteLine(current.ToString("d. M. HH:mm:ss (gg)"));
Console.WriteLine(current.ToString("dd-MM-yyyy-hh-mm-ss-z"));
```

Při práci s časem si je potřeba dávat pozor na to, jestli můžeme danému datu a času věřit. Pokud používáme funkce předpřipravené v C#, nebudeme mít ani problém s přesuny hodin a ani pásmy.

Čtení znaku z konzole

Při čtení z konzole můžeme zavolat metodu `Console.ReadKey()`, která nám přečte následovně zmáčknuté tlačítko. Z daného tlačítka můžeme např. číst i to, zda uživatel zmáčkl nějaké pomocné tlačítko, jako control či podobné.

```
ConsoleKeyInfo keyInfo = Console.ReadKey();

ConsoleKey key = keyInfo.Key;

if(key == ConsoleKey.K) {
    Console.WriteLine("Utečte!");
    break;
}

if(keyInfo.Modifiers.HasFlag(ConsoleModifiers.Control))
    Console.WriteLine("proč mačkáš K s kontrolem?");

Console.ReadLine();
```