**Datové struktury s klíčem, indexem a hashované**

**S indexem (index)**

* Array
* List
* ArrayList

**Array**

* V datové struktuře pole můžeme uložit více proměnných stejného datového typu
* Deklarujeme pole zadáním datového typu jeho prvků
* Chceme-li, aby pole ukládalo prvky libovolného typu, můžeme jako datový typ zadat object
* Pole je v podstatě hromada krabic poskládaných vedle sebe a k tomu očíslovaných přirozenými čísly začínající 0
* Array je statická datová struktura, tzn. že za běhu programu nemůže být modifikována

**List**

* List je oproti poli dynamická datová struktura, můžeme ho tedy měnit za běhu programu, přidávat, ubírat, ptát se na jeho velikost atd…
* Navíc lze List převést do pole toArray()
* Jde jen o to, že list zkopíruje všechny své prvky do nového pole se stejnou velikostí jako má list
* List je chytřejší, dynamické pole, tzn. interně dělá to, co jsme si popsali výše u pole a co jsme většinou líní implementovat ručně
* Všechny časové složitosti, týkající se vyhledávání, vložení, smazání atd. jsou stejné jako u pole
* Výhoda je však ta, že místo složité syntaxe, kterou jsme museli dříve sami psát stačí zavolat námi zvolenou metodu
* List.Add(), List.Delete(), List.Sort(), List.Contains()...
* List je vnitřně implementován pomocí pole, při přidání nového záznamu se tedy při vyčerpání kapacity vytvoří interně pole nové a prvky se do něj zkopírují
* List je tzv. generická kolekce, při deklaraci Listu musíme specifikovat datový typ, který v něm bude uložený

**ArrayList**

* ArrayList je non-generic kolekce, tzn. lze do něj uložit jakýkoli datový typ
* Elementy můžou být přidány nebo odebrány v jakémkoliv čase (i za běhu programu oproti Array)
* Povoluje duplicitní hodnoty
* Není nadále doporučeno používat ArrayList, místo toho je doporučen List<T>
* List může narozdíl od ArrayListu pracovat s LINQ

**Hashované**

* Hashtable
* HashSet

**Hashtable**

* Hashtable ukládá data jako key-value pairs (klíč / hodnota), které jsou uspořádány v závislosti na “heši” (hash) klíče
* Hash tabulka optimalizuje vyhledávání výpočtem hashového kódu každého klíče a uloží jej do jiného bloku interně a poté odpovídá hašovacímu kódu zadaného klíče v okamžiku přístupu k hodnotám
* Tato kolekce je non-generic (negenerická), tzn. při vytvoření instance se nezadává jaký datový typ bude ukládat, můžeme to kombinovat
* Kolekce je v namespace System.Collections
* Doporučeno již nepoužívat a namísto toho používat Dictionary<TKey, TValue>
* Stejně jako u ArrayList nemáme “type safety”, tzn. že tam můžeme vložit random typ (kompilátoru to nevadí, ale při např. procházení foreachem dojde k runtime errorům)
* Nemusíme castit (konvertovat) hodnoty, které si z kolekce vezmeme

**Důležité body:**

* Klíč nemůže být null, ale hodnota ano
* Klíč musí být unikátní, duplicitní klíče nejsou povoleny
* Objekt klíče musí být neměnný tak dlouho, dokud je použit jako klíč
* V této kolekci můžeme ukládat prvky jakéhokoli datového typu
* Hash funkce je poskytována každým objektem klíče v Hashtable
* Hashtable získává prvky tím, že porovnává “heš” kódy (hashcode) klíčů, je to pomalejší v provedení než Dictionary
* Hashtable používá defaultní hashcode provider, což je object.GetHasCode(), ovšem můžeme využít i vlastní hashcode provider

**HashSet**

* Generická kolekce
* Nelze ukládat duplicitní elementy
* Tato kolekce není v seřazeném pořadí a ani ji nelze následně seřadit pomocí metody
* HashSet je dynamická kolekce, což znamená, že velikost HashSetu se automaticky zvyšuje při přidání nových elementů
* Nachází se v System.Collections.Generic
* HashSet je vlastně Dictionary<TKey, TValue> , ale můžeme udávat pouze hodnotu a klíč se vygeneruje sám
* Pro ukládání používá hash tabulku stejně jako Dictionary a pracuje i na stejném principu
* V porovnání s Listem má mnohem větší rychlost (např. díky vyhledávání pomocí hashe)

**S klíčem (key / value) - generické datové typy**

* Dictionary
* SortedDictionary
* ConcurrentDictionary
* ListDictionary
* SortedList

**Dictionary**

* V C # je Slovník obecná kolekce, která se obvykle používá k ukládání párů klíč / hodnota (key / value)
* Výhodou slovníku je, že jde o generický typ
* Slovník je definován v System.Collection.Generic namespace
* Jeho dynamický charakter znamená, že velikost slovníku roste podle potřeby
* Na haldě je uložen rozházeně, tzn. není uložen v jednom bloku
* Prvky jsou rozmístěné po haldě, nejsou pohromadě jako u Listu
* Ve slovníku nemůže být klíč nulový, ale hodnota být nulová může
* Ve slovníku musí být klíč jedinečný
* Duplicitní klíče nejsou povoleny, pokud se pokusíte použít duplicitní klíč, kompilátor vyvolá výjimku
* Ve slovníku můžeme ukládat pouze stejné typy prvků

**SortedDictionary**

* Používá k reprezentaci kolekce párů klíč / hodnota
* Tento pár je v seřazené podobě a seřazení se provádí na klíči, ovšem je seřazen až v době užití, jinak je uložený rozházeně na haldě
* Klíče jsou neměnné, vždy jedinečné a nemohou být null
* Pokud je typ hodnoty referenčního typu, můžeme použít hodnotu null, hodnota nemusí být jedinečná
* Třída SortedDictionary poskytuje nejrychlejší operace vkládání a odebírání netříděných dat
* Pár klíč / hodnota třídy SortedDictionary je získán pomocí struktury KeyValuePair

**ConcurrentDictionary**

* Je kolekce bezpečná pro vlákna k ukládání párů klíč / hodnota
* ConcurrentDictionary lze použít s více vlákny současně
* Práce s ConcurrentDictionary je thread-safe až na metody AddOrUpdate() nebo GetOrAdd()
* Bez třídy ConcurrentDictionary, pokud musíme použít třídu Dictionary s více podprocesy, musíme použít zámky k zajištění bezpečnosti podprocesů, která je vždy náchylná k chybám
* ConcurrentDictionary nám poskytuje snadné řešení místo toho
* Interně spravuje zamykání a poskytuje snadné rozhraní pro přidávání / aktualizaci položek

**ListDictionary**

* Non-generic kolekce
* Specializovaná kolekce, která se nachází v namespace System.Collections.Specialized
* Je implementován s jednosměrným linked listem
* Odlehčená verze Dictionary kolekce, implementuje IDictionary, její metody a vlastnosti
* Je doporučena k použití pouze při menším počtu prvků než je 10
* Elementy nejsou v žádném zaručeném pořadí (náhodné)
* Klíč nemůže být null, hodnota ano

**SortedList**

* Při vytváření určujeme zda-li to bude non-generic (System.Collection) nebo generic (System.Collection.Generic)
* Prvky v této kolekci se řadí
* Generický se chová jako Dictionary jen se řadí
* Negenerický je takový pozůstatek (dědeček) .NETu, vytvořen při verzi 1.1 a je tam jen kvůli zpětné kompatibilitě
* Negenerický se chová taky tak s výjimkou toho, že LZE přistupovat k prvkům (values) pomocí indexu

(To u generického NEPLATÍ!!)

* U negerického můžeme proměňovat jednotlivé datové typy VALUES ale u KEY to tak neplatí
* KEY musí být vždy jedné datové struktury, jinak compiler vyhodí výjimku