Spring4D Collections and O() Complexity

Primož Gabrijelčič

ROME - NOVEMBER 14th, 15th 2024

About me

Primož Gabrijelčič

http://primoz.gabrijelcic.org

- programmer, MVP, writer, blogger, consultant, speaker
- Blog *https://thedelphigeek.com*
- Twitter *@thedelphigeek*
- Skype *gabr42*
- LinkedIn *gabr42*
- GitHub *gabr42*
- SO *gabr*

The Delphi Geek

random ramblings on Delphi, programming, Delphi programming, and all the rest

Friday, April 05, 2024

Delphi and Python, working happily together

Next Wednesday, 10th, I'll be talking about Delphi and Pyhon in Ljubljana. As usual for Slovenian workshops, the talk will be in Slovenian language so I'll continue this invitation appropriately ...

Python je izjemno priljubljen programski jezik, ki slovi po svoji raznoliki in preprosti uporabi. Skupnost razvijalcev nenehno prispeva k njegovemu razvoju in izboljšanju jezika, kar dodatno krepi njegovo privlačnost za širok nabor uporabnikov. Kot večina večplatformnih sistemov pa ne slovi po podpori za izdelavo uporabniških vmesnikov.

Tu nastopi **Delphi**, ki slovi po svoji hitrosti in učinkovitosti in je odličen za razvoj aplikacij za Windows okolje. Zato ni presenetljivo, da okolji ne delujeta kot neposredna konkurenca, pač pa se izredno dobro **dopolnjujeta**.

Največja prednost je združevanje moči obeh jezikov. Delphi ponuja hitrost in zmogljivost, Python pa berljivost in obsežno knjižnico razširitev. **S tem lahko razvijalci izkoristite najboljše iz obeh svetov.**

Na predavanju si bomo ogledali:

- Kako lahko Python in Delphi sodelujeta na dva različna načina,
- predstavili bomo orodji DelphiVCL in DelphiFMX, s katerima lahko v Pythonu enostavno ustvarite lep uporabniški vmesnik
- ter nekaj orodij (Python4Delphi, PythonEnvironments, Lightweight Python Wrappers), ki omogočajo, da kodo, napisano v Pythonu, vgradite v svoj Delphi program.

Spotoma si bomo na kratko ogledali še novosti v novem **Delphi 12.1**!

Kliknite za prijavo!

Posted by gabr42 at 09:39 No comments:

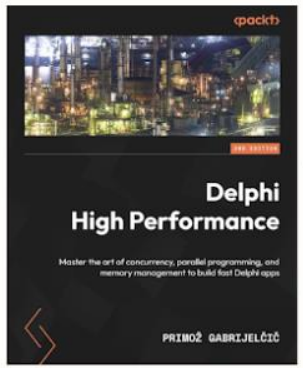
Labels: Delphi, presentations, Python

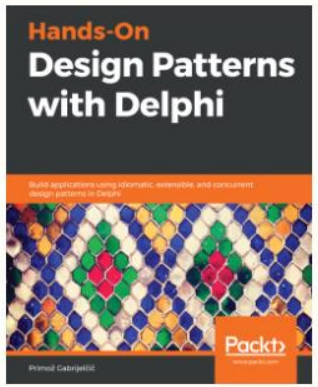
Saturday, December 02, 2023

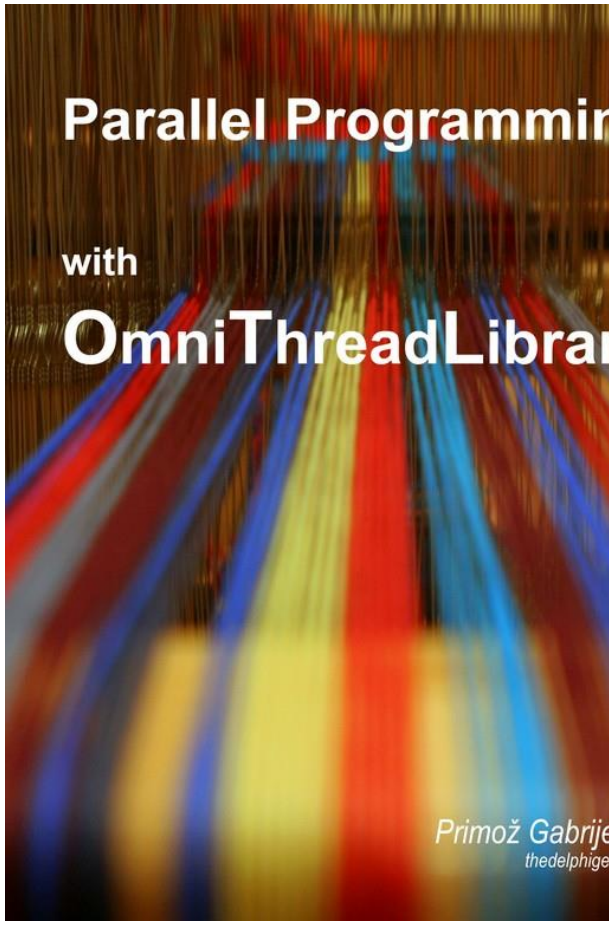
embarcadero MVP

Pages

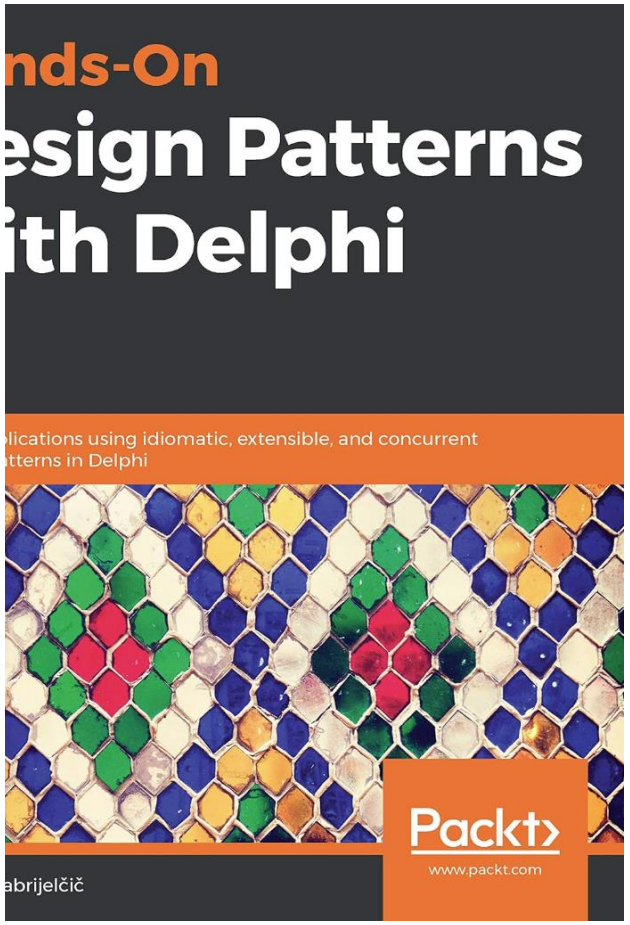
Presentations







https://delphi-books.com/



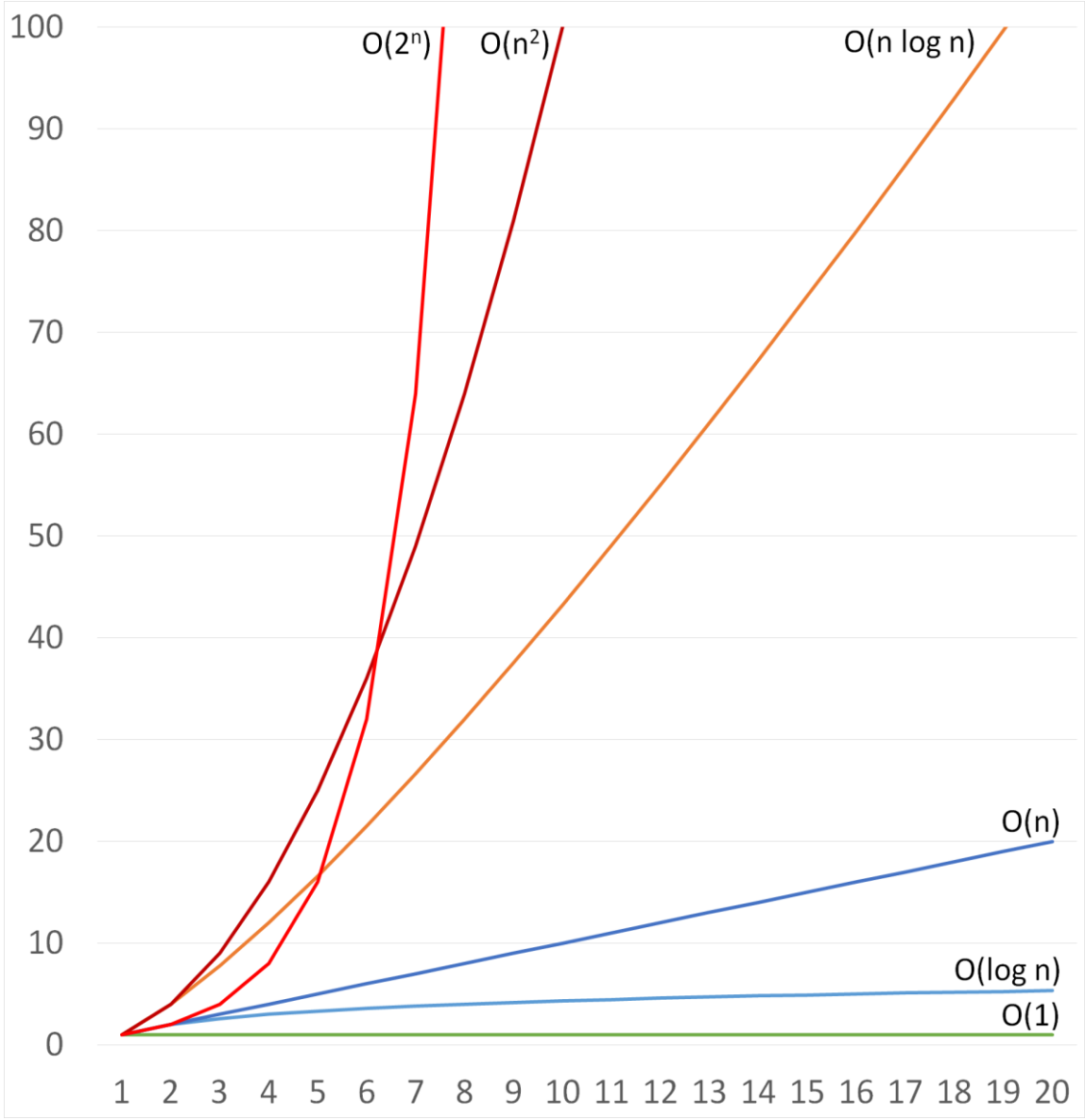
Algorithm complexity

Algorithm complexity

- Tells us how algorithm slows down if data size is increased by a factor of *n*
- O()
  - O(n), O(n^2), O(n log n) …
- Time **and** space complexity
- Constant factors are ignored …
  - … but can be important

Examples

- O(1) accessing array elements
- O(log n) searching in ordered list
- O(n) linear search
- O(n log n) quick sort (average)
- $O(n^2)$ quick sort (worst), naive sort (bubblesort, insertion, selection)
- $O(c^n)$ recursive Fibonacci, travelling salesman



Comparing complexitie

Data size	O(1)	n^2)	O(c^n)
1	1		1
10	1	)	512
100	1	000	10^{29}
300	1	000	10^{90}

http://bigocheatsheet.com/

Spring4D Collections

About Spring4D

- Collection of many nice things
  - Smart pointers, nullable values, multi-dispatch events ...
  - Dependency Injection container
  - Mocking
  - RTTI improvements
  - ...
  - **Collections**
- Home
  - https://bitbucket.org/sglienke/spring4d
- License: Apache 2.0
  - https://www.tldrlegal.com/license/apache-license-2-0-apache-2-0
- Installation
  - Download
  - Unzip
  - Add to Search path

Spring4D Collections

- Lists
- Stacks & queues
- Sets & bags
- Dictionaries
- Multimaps
- (Trees)
- ...

Important Spring4D Collection Interfaces

- Rich API
- **IEnumerable<T>**
  - IStack<T>
  - IQueue<T>
  - **ICollection<T>**
    - IList<T>
    - ISet<T>
    - IMultiSet<T>
    - IMap<K,V> (= ICollection<TPair<K,V>>)
      - IDictionary<K,V>
      - IMultimap<K,V>

Spring4D Collections Data Storage

- **Array**-based
  - List, sorted list, queue, stack
  - Access O(1), Search O(n)/O(log n), Insert O(n)/O(1), Delete O(n)/O(1)
- **Tree**-based
  - Sorted set, sorted dictionary, sorted multimap
  - Red-black (balanced) tree, allocated as a dynamic array of dynamic arrays
  - Access O(log n), Search O(log n), Insert O(log n), Delete O(log n)
- **Hash table**-based
  - Unsorted set, unsorted dictionary, unsorted multimap
  - Python 3 hash table, allocated as a dynamic array
  - Elements are ordered in insertion order
  - Access O(1) (*O(n)), Search O(1), Insert O(1), Delete O(1) (*)
  - O(1) operations, but slower than List O(1)

IList

- IList<T>
- Simple array of values
- Over-allocating, automatic copy on overflow
  - Just like Delphi implementation
- Access
  - O(1) Fastest possible access, just dereference a pointer
- Search
  - O(n) Worst case: must check all elements
- Insert
  - O(n) Average case, must move up to all elements
  - O(1) Best case, at the end
- Delete
  - O(n) Average case, must move up to all elements
  - O(1) Best case, at the end

Sorted IList<T>

- Works like TStringList with Sorted := True
- Keeps order when adding elements
- IndexOf uses binary search
- Different than TList<T>
  - Must call Sort explicitely
  - Must use BinarySearch instead of IndexOf
- Search
  - O(log n) Bisection

IStack<T> and IQueue<T>

- Fast because of simplified interface
- Insert, Delete
  - O(1) Almost always
  - O(n) If storage needs to grow
- Stacks
  - Normal
  - Bounded
- Queues
  - Normal
  - Bounded
  - Evicting
- Deque
  - Double-ended queue

ISet<T>, IMultiSet<T>

- Set
  - A collection of elements, each value can appear at most once
  - Set operations (union, intersection, subset, superset …)
  - Either a hash table (unsorted) or tree (sorted)
- Multiset
  - “Bag“
  - Element + counter
  - Either a hash table (unsorted) or tree (sorted)
- Access = Search
  - O(1) (unsorted) / O(log n) (sorted)
- Insert
  - O(1) (unsorted) / O(log n) (sorted)
- Delete
  - O(1) (unsorted) / O(log n) (sorted)

IDictionary<K,V>

- “Array[K] of T”
- Rewritten in Spring4D v2
- Either a hash table (unsorted) or tree (sorted)
- Access (key)
  - O(1) (unsorted) / O(log n) (sorted)
- Search (value)
  - O(n)

- IBiDiDictionary<K,V>
  - Two interconnected dictionaries
  - Always unsorted
  - Search (value)
    - O(1)

IMultimap<K,V>

- A dictionary that maps values into:
  - Lists (CreateMultiMap)
    - Unordered values, can contain duplicates
  - Unsorted sets (CreateHashMultiMap)
    - Unordered values, no duplicates
  - Sorted sets (CreateTreeMultiMap)
    - Ordered values, no duplicates
- In addition, keys can be unsorted (hash table) or sorted (tree)
- O() for keys is the same as for IDictionary
- O() for values is the same as for List / Set / Sorted set

Use Cases

Use Cases: List, Stack, Queue

- List, sorted list
  - Small number of elements
    - CPUs are fast!
  - Storage and retrieval
  - (Searching when sorted)
    - Slower element insertion
    - There are more appropriate collections
  - As a replacement for TList
- Stack
  - When your algorithm needs a stack
- Queue
  - When your algorithm needs a queue

Use Cases: Set, MultiSet

- Set
  - When you want to store elements and later check whether they are there
  - Basically any time an algorithm calls for a hash table (unsorted set) or a tree (sorted set)
- Multiset
  - When you want to store elements and also count them
  - Could be used to implement some reference-counted mechanism, for example
- Unsorted
  - Most of time
- Sorted
  - Only if you need to enumerate [multi]set (or parts of it) in order

Use Cases: Dictionary, MultiMap

- Dictionary
  - As a replacement for TDictionary
  - For mapping one set of values into another
  - If you don’t need the “value” part, use a Set
- BiDiDictionary
  - When you need fast bidirectional mapping between two sets of values
- MultiMap
  - When mapping one set of values into collections of values
  - Value → list, value → set, value → sorted set

My Code

- List 53%
- Dictionary 25%
- Set 6%
- SortedList 6%
- ObjectList 3%
- SortedDictionary 3%
- SortedSet 1%
- BidiDictionary 1%
- MultiMap 1%
- Queue 0.5%
- SortedObjectList 0.5%
- EvictingQueue 0.5%

Parting Thoughts

Keep In Mind

- Select proper collection for your problem
  - Changing the algorithm (e.g. picking a more appropriate collection) is always better than fine-tuning the code
- Learn the collections API (starting from IEnumerable<T>)
- Custom solutions can be made by combining collections (BidiDictionary, MultiMap)
- Constant factor is important – don’t complicate if you have only small number of elements and infrequent access
  - List with 10 elements will beat any other collection type
- If you need performance, measure!