CIS 200 Final exam class notes

Ch 23 – all of the iList and iEnumerable, etc.. don’t need to know the details about those

* Collection classes store collections of data
  + each instance of one of these classes is a collection of items

Ch22 - what advantage does providing a generic class over a non generic class (Like in 21) that uses an object as the data type

* Auto inboxing – not downcasting before adding to the array
  + Compile time type safety enforcement - detects type mismatches at Compile time
* Generics – provide compile time safety
  + Generic Methods enable you to specify with a method declaration, a set of related methods
  + Generic Classes enable you to specify with a single class declaration, a set of related classes
  + Generic interfaces enable you to specify, with a single interface declaration, a set of related interfaces.
  + All generic method declarations have a **type parameter list** delimited by angle brackets that follows the method’s name.
    - A type parameter is can be used to declare the return type, the parameter types and local variable types in a generic method declaration.
    - Private static void **displayArray< T >**(T[] input array)
  + Type parameters act as place holders for **Type arguments,** that represent the types of data that will be passed to the generic method
  + **Type inferencing** is the process of inputting a type in the method for the type argument (T) such as an INT.
* Type constraints
  + IComparable<T> interface
    - It is possible to compare two objects of the same type if that type implements the generic interface IComparable. These objects can be sed with the sorting and searching methods we learned about.
      * Used the method compareTo(XXX);
  + **Where claus**e specifies the type constraint for type parameter T
    - Private static void **displayArray< T >**(T[] input array) where T : IComparable<T>
      * This indicates this method requires the type argument to implement interface IComparable<T>. If NO type constraint is given then the default is OBJECT
    - A **Class constraint** indicates that the type argument must be an object of a specific base class or one of its subclasses
    - An **Interface constraint** indicates that the type argument’s class must implement a specific interface
    - It is possible to apply multiple constraints to a type parameter by providing commas in the where clause
  + Overloading
    - Generic methods CAN be overloaded

Be able to take an existing method and turn it into a generic method.

Be able to write a generic class

CH21 – compare and contrast

* Linked lists are collections of data items “lined up in a row” users can make insertions and deletions anywhere in the list
  + Linear collection of self-referential class objects, called nodes, connected by reference links.
    - A program accesses a linked list via the reference to the first node
* Insertions and deletions are make at the top of a stack
* Queues are inserted at the back (AKA the tail) and deletions are made in the front (AKA the head)
* Binary trees facilitate high speed searching and sorting of data, efficient elimination of duplicated data items, representation of file system directories and compilation of expressions into machine language.
* Single linked list – inserting in front(Constant), insert in back(constant), moving from front(constant), moving from back (Linear). What are the performance characteristics? (Constant or linear)
* Doubly linked list - performance characteristics?
  + Allows traversals both forward and backward. Such a list is often implemented with two start references – one that refers to the first element of the list to allow front to back traversal of the list and one that refers to the last element to allow back to front traversals .
  + Each one has a reference to the node in front and behind it.
* Circular doubly linked list
  + The first node’s reverse reference refers to the last node and the last nodes forward reference refers to the first node.
* Stack – LIFO (last in first out)
  + Is a constrained version of a linked list – it receives new nodes and releases nodes only at the top.
  + Stacks support recursive method calls in the same manner that they do conventional non recursive method calls
  + Push – Adds a new node to the top of the stack
  + Pop - removes a node from the top of the stack
* Queue – FIFO (first in first out)
  + Similar to a checkout line in a supermarket.
  + Computers with only one processor can only do one thing at a time so it would perform in queues
  + **Print Spooling** – when the a single printer is connected to many users the jobs are sent to a queue until they are performed and de queued.
  + **Spooler** – a computer program used to manage the queue to ensure that as each print job completes, the next one is sent to the printer.
  + En queue – insert
  + De queue – remove
* Structs - store object references
  + Type declared with the keyword struct are implicitly value types
  + Boxing conversion
    - Enables simple types to be used anywhere objects are expected. The simple type is copied into an object so that the simple type value can be manipulated as an object.
  + Un-boxing
    - Can be used to explicitly convert an object reference to a simple value
  + Self referential classes
    - Contains a reference member that refers to an object of the same class type.
    - These self-referenced objects can be linked together to make useful data structures such as links, lists, queues, stacks and trees.
  + Dynamic memory allocation
    - A program’s ability to obtain more memory space at execution time to hold new nodes and to release space no longer needed
* Trees are non-linear, two-dimensional data structures with special properties. Tree nodes can contain two or more nodes

Binary tree

Binary search tree

* Storage rules
  + Left is less than or equal to the node
  + Greater than the node
* Be able to build a binary search tree from scratch

Traversals

* Pre –
  + Traverse the node
  + Left
  + Right
* Post –
  + Left
  + Right
  + Node
* In order –
  + If null do not process tree
  + Traverse the left sub tree
  + Traverse the node
  + Traverse the right

Given a tree perform the traversals

Extra credit – given the traversals create the tree (part 4)

CH20 – New part of chapter 20

* Sorts – simple knowledge, a description of a problem and choose which sort to choose (Essay with details of why the others are the wrong choice)
  + Selection – the first algorithm selects the smallest element in the array and swaps it with the first element. The second iteration selects the second smallest and swaps it with the second element in the array. Recursion until the array is finished (
  + Merge – recursive – wont do a complete sort but maybe a complete pass
  + Quick – recursive – wont do a complete sort but maybe a complete pass
  + Insertion – (Linear best case)
* Be able to draw out the sorts on paper step by step
* Know the performance characteristic of each and be able to choose the difference in them all.