

# OBJECTIVE-C FOUNDATION CLASSES

## REFERENCE CARD

### Part 2: Collections

---

#### DArray

##### Methods

- **init** ..... Init empty array
- **init :(long) length** ..... Init array with length
- **copy** ..... (Deep)copy of the object
- **shallowFree** ..... Free the array, not the objects
- **free** ..... Free the array and the objects
- **(BOOL) isValid :(long) index** ..... Test if index is valid
- **(long) length** ..... Get length of array
- **length :(long) length** ..... Increase length of array
- **(id) set :(long) index :(id) obj** ..... Set object in array
- **(id) get :(long) index** ..... Get object in array
- **(long) count :(id) obj** ..... Count occurrences of object
- **(long) index :(id) obj** ..... Smallest index for object
- **(long) rindex :(id) obj** ..... Greatest index for object
- **(BOOL) has :(id) obj** ..... Test presence object
- **each :(SEL) sel** ..... Perform sel for every object

#### DAvLTree

##### Methods

- **init :(Class) key** ..... Init empty tree with class key
- **free** ..... Free the tree and the stored objects
- **shallowFree** ..... Free the tree, not the objects
- **(BOOL) isEmpty** ..... Test for empty table
- **(long) length** ..... Return the number of objects
- **(id) delete :(id) key** ..... Delete key,object from the tree
- **get :(id) key** ..... Return the object related to a key
- **(BOOL) has :(id) key** ..... Check if the tree has a key
- **(BOOL) insert :(id) key :(id) obj** ..... Insert key,object
- **(DList \*) keys** ..... Return a list with (sorted) copied keys
- **(DList \*) objs** ..... Return a list with copied objects
- **(DList \*) rkeys** Return a list with (rev. sorted) copied keys

#### DAvLIterator

##### Methods

- **init** ..... Init empty iterator
- **init :(DAvLTree \*) tree**
  - └ Init iterator with tree, move to root
- **tree :(DAvLTree \*) tree** .... Set the avl tree, move to root

- **(id) first** ..... Get first object
- **(id) next** ..... Get next object
- **(id) prev** ..... Get previous object
- **(id) last** ..... Get last object
- **(id) key** ..... Get key in current node
- **(id) object** ..... Get object in current node
- **(id) object :(id) obj** ..... Set object in current node

#### DCircle

##### Methods

- **init** ..... Init circular buffer with size 1
- **init :(long) size** ..... Init circular buffer with size
- **copy** ..... Copy the buffer (with the objects)
- **shallowFree** ..... Free the buffer, not the objects
- **free** ..... Free both the buffer and the objects
- **(long) size** ..... Return the size of the buffer
- **size :(long) size** ..... Enlarge size of the buffer
- **(long) length** ..... Return the number of objects
- **(BOOL) isEmpty** ..... Test for empty buffer
- **(BOOL) isFull** ..... Test for full buffer
- **(BOOL) isValid :(long) index** ..... Check for valid index
- **(id) get :(long) index** ..... Get indexth object
- **(BOOL) push :(id) obj** ..... Push object on stack
- **(id) pop** ..... Pop object from stack
- **(id) tos** ..... Return object on top of stack
- **(BOOL) enqueue :(id) obj** ..... Put object in queue
- **(id) dequeue** ..... Get object from queue
- **each :(SEL) sel** ..... Perform sel from start till last
- **reach :(SEL) sel** ..... Perform sel from last till start

#### DCube

##### Methods

- **init** ..... Init empty cube
- **init :(int) cols :(int) rows :(int) layers**
  - └ Init cube with size
- **copy** ..... (Deep)copy of the object
- **shallowFree** ..... Free the cube, not the objects
- **free** ..... Free the cube and the objects
- **(BOOL) isValid :(int) c :(int) r :(int) l**
  - └ Is row/col/layer valid?
- **(long) length** ..... Get length of cube
- **(int) columns** ..... Get number of columns
- **columns :(int) cols** ..... Increase number of columns

- **(int) rows** ..... Get number of rows
- **rows :(int) rows** ..... Increase number of rows
- **(int) layers** ..... Get number of layers
- **layers :(int) layers** ..... Increase number of layers
- **(id) set :(int) c :(int) r :(int) l :(id) obj**
  - └ Set object in cube
- **(id) get :(int) c :(int) r :(int) l**. Get object from cube
- **(long) count :(id) obj** ..... Count occurrences of object
- **(BOOL) has :(id) obj** ..... Test presence object
- **each :(SEL) sel** ..... Perform sel for every object

#### DGraph

##### Methods

- **init** ..... Init empty graph
- **init :(char \*) attributes** ..... Init graph with attributes
- **shallowFree** ..... Free graph without the stored objects
- **free** ..... Free graph with the stored objects
- **(char \*) attributes** ..... Return the graph attributes
- **attributes :(char \*) attributes** ..... Set the attributes
- **(DGraphNode \*) addNode :(char \*) label :(char \*) attr**
  - └:(id) object ..... Add a new node to the graph
- **(BOOL) addNode :(DGraphNode \*) node** ... Add node to graph
- **(DGraphEdge \*) addEdge :(char \*) label :(char \*) attr**
  - └:(double) weight :(id) object :(DGraphNode \*) source
  - └:(DGraphNode \*) target ..... Add a new edge to the graph
- **(BOOL) addEdge :(DGraphEdge \*) edge**
  - └:(DGraphNode \*) source :(DGraphNode \*) target
  - └ Add an edge to the graph
- **(BOOL) reroute :(DGraphEdge \*) edge**
  - └:(DGraphNode \*) source :(DGraphNode \*) target
  - └ Reroute an existing edge
- **(DList\*) shortestPath :(DGraphNode\*) from**
  - └:(DGraphNode\*) to ..... Determine shortest path in graph
- **(BOOL) hasNode :(DGraphNode \*) node** .... Check for a node
- **(BOOL) hasEdge :(DGraphEdge \*) edge** ... Check for an edge
- **(id) removeNode :(DGraphNode \*) node**
  - └ Remove an unconnected node
- **(id) removeEdge :(DGraphEdge \*) edge** .... Remove an edge
- **(DListIterator \*) nodes**
  - └ Return an iterator on the graph nodes
- **(DListIterator \*) edges**
  - └ Return an iterator on the graph edges
- **(BOOL) toDot :(id <DTextWritable>) writer**
  - └ Export to dot file

## DGraphEdge

### Methods

- **init** ..... Init empty edge
- **init** : (char \*) **label** : (double) **weight** : (id) **object**
  - | Init an edge with a label, weight and object
- **init** : (char \*) **label** : (char \*) **attributes**
  - | : (double) **weight** : (id) **object**
    - | Init an edge with label, weight, attr. and object
- **shallowFree** ..... Free the edge without the stored object
- **free** ..... Free the edge and the stored object
- (char \*) **name** ..... Return the name of the edge
- **name** : (char \*) **name** ..... Set the name of the edge
- (char \*) **label** ..... Return the label of the edge
- **label** : (char \*) **label** ..... Set the label of the edge
- (double) **weight** ..... Return the weight of the edge
- **weight** : (double) **weight** ..... Set the weight of the edge
- (char \*) **attributes** ..... Return the edge attributes
- **attributes** : (char \*) **attributes** ..... Set the attributes
- (id) **object** ..... Return the stored object
- (id) **object** : (id) **object** ..... Set the stored object
- **connect** : (DGraphNode \*) **source** : (DGraphNode \*) **target**
  - | Connect source with target by the edge
- **disconnect** ..... Disconnect the edge
- (DGraphNode \*) **source** ..... Return the source node
- (DGraphNode \*) **target** ..... Return the target node
- **reverse** ..... Reverse the egde

## DGraphNode

### Methods

- **init** ..... Init empty graph node
- **init** : (char \*) **label** : (id) **object**
  - | Init node with label and object
- **init** : (char \*) **label** : (char \*) **attributes** : (id) **object**
  - | Init node with label, attributes and object
- **shallowFree** ..... Free the node without the stored object
- **free** ..... Free the node and the stored object
- (char \*) **name** ..... Return the name of the graph node
- **name** : (char \*) **name** ..... Set the name of the node
- (char \*) **label** ..... Return the label of the node
- **label** : (char \*) **label** ..... Set the label of the node
- (char \*) **attributes** ..... Return the attributes
- **attributes** : (char \*) **attributes** ..... Set the attributes
- (id) **object** ..... Return the stored object
- (id) **object** : (id) **object** ..... Set the stored object

- **path** ..... Reset the path
- **path** : (DGraphNode \*) **prev** : (double) **sum** .... Set the path
- **prev** ..... Return the previous graph node in the path
- (double) **sum** ..... Return the weight sum in the path
- (unsigned long) **ingoingDegree** ..... Return ingoing degree
- (unsigned long) **outgoingDegree** .... Return outgoing degree
- (unsigned long) **degree** ..... Return degree of node
- **addIngoingEdge** : (DGraphEdge \*) **edge** .... Add ingoing edge
- (BOOL) **removeIngoingEdge** : (DGraphEdge \*) **edge**
  - | Remove ingoing edge
- **addOutgoingEdge** : (DGraphEdge \*) **edge** .. Add outgoing egde
- (BOOL) **removeOutgoingEdge** : (DGraphEdge \*) **edge**
  - | Remove outgoing edge
- (DListIterator \*) **ingoingEdges**
  - | Return ingoing edges iterator
- (DListIterator \*) **outgoingEdges**
  - | Return outgoing edges iterator

## DHashTable

### Methods

- **init** : (Class) **key** ..... Init table with key class
- **init** : (Class) **key** : (ulong) **size** : (double) **load**
  - | Init table with key class, start size and load factor
- **shallowFree** ..... Free the table, not the objects
- **free** ..... Free the table and the objects
- (long) **length** ..... Return number of objects in table
- (BOOL) **isEmpty** ..... Test for empty table
- (double) **load** ..... Return the load factor
- **load** : (double) **load** ..... Set the load factor (0.0)
- (ulong) **size** ..... Return the size of the table
- **size** : (ulong) **size** ..... Set the size of the table
- (BOOL) **insert** : (id) **key** : (id) **obj** ... Insert object in table
- (BOOL) **delete** : (id) **key** ..... Delete object from table
- (BOOL) **has** : (id) **key** ..... Test for presence in table
- (id) **get** : (id) **key** ..... Return object in table
- **each** : (SEL) **sel** ..... Perform method for every object
- (DList \*) **keys** ..... Return new list with copied keys
- (DList \*) **objects** ..... Return new list with copied objects

## DHashIterator

### Methods

- **init** ..... Init empty table iterator
- **init** : (DHashTable \*) **table** ..... Init table iterator

- **hashTable** : (DHashTable \*) **table** ..... Set table for iterator
- (id) **first** ..... Go to first object in table
- (id) **next** ..... Go to next object in table
- (id) **prev** ..... Go to previous object in table
- (id) **last** ..... Go to last object in table
- (id) **object** : (id) **obj** ..... Set object in current node
- (id) **object** ..... Get object from current node
- (id) **key** ..... Get key for object from current node

## DList

### Methods

- **init** ..... Init an empty list
- **shallowFree** ..... Free the list, not the objects
- **free** ..... Free the list and the objects
- (BOOL) **isEmpty** ..... Test for empty list
- (long) **length** ..... Get number elements in list
- **prepend** : (id) **obj** ..... Prepend object in list
- **append** : (id) **obj** ..... Append object in list
- (BOOL) **push** : (id) **obj** ..... Push object at end of list
- (id) **pop** ..... Pop the last object in the list
- (id) **tos** ..... Return last object in list
- (BOOL) **enqueue** : (id) **obj** . Enqueue the object at end of list
- (id) **dequeue** ..... Dequeue the first object in the list
- **insert** : (long) **index** : (id) **obj**
  - | Insert object indexed in list
- (id) **set** : (long) **index** : (id) **obj** .... Set the object in list
- (id) **get** : (long) **index** ..... Get the object in list
- (DList \*) **get** : (long) **from** : (long) **to**
  - | Get new list with ranged objects (references)
- (id) **delete** : (long) **index** .. Delete indexed object from list
- (DList \*) **delete** : (long) **from** : (long) **to**
  - | Remove ranged objects to new list
- (long) **count** : (id) **obj** ..... Count the object in list
- (long) **index** : (id) **obj** ..... Smallest index for object
- (long) **rindex** : (id) **obj** ..... Biggest index for object
- (BOOL) **has** : (id) **obj** ..... Test if object in list
- (BOOL) **remove** : (id) **obj** ..... Remove object from list
- **reverse** ..... Reverse sequence of objects in list
- **each** : (SEL) **sel** ..... Perform method for every object
- + (DList \*) **split** : (char \*) **str** : (char) **sep** : (int) **max**
  - | Split string in list of strings
- (DText \*) **join** : (char) **sep** .... Join list of strings to string

## DListIterator

### Methods

- **init** ..... Init iterator without list
- **init** : (DList \*) list ..... Init iterator with list
- **list** : (DList \*) list ..... Set new list for iterator
- (id) **first** ..... Go to first object in list
- (id) **next** ..... Go to next object in list
- (id) **prev** ..... Go to previous object in list
- (id) **last** ..... Go to last object in list
- (id) **object** : (id) obj ..... Set object in current node
- (id) **object** ..... Get object from current node

## DSortedList : DList

### Methods

- **init** ..... Init empty ascending sorted list
- **init** : (Class) class : (BOOL) ascending ..... Init sorted list
- **shallowFree** ..... Free the list, not the objects
- **free** ..... Free both list and stored objects
- (BOOL) **ascending** ..... Is the list sorted ascending
- **ascending** : (BOOL) ascending ..... Set sorting direction
- (Class) **class** ..... Return the class of the objects
- **class** : (Class) class ..... Set the class of the objects
- **insert** : (id) object ..... Insert an object, sorted
- **reverse** ..... Reverse the list
- + (DSortedList \*) **splitSorted** : (char \*) str : (char) sep  
    └ : (int) max ..... Split string in list of sorted strings

## DTable

### Methods

- **init** ..... Init empty table
- **init** : (int) cols : (int) rows ..... Init table with size
- **copy** ..... (Deep)copy of the object
- **shallowFree** ..... Free the table, not the objects
- **free** ..... Free the table and the objects
- (BOOL) **isValid** : (int) c : (int) r ..... Is row/col valid?
- (long) **length** ..... Get length of table
- (int) **columns** ..... Get number of columns
- **columns** : (int) cols ..... Increase number of columns
- (int) **rows** ..... Get number of rows
- **rows** : (int) rows ..... Increase number of rows
- (id) **set** : (int) c : (int) r : (id) obj .. Set object in table
- (id) **get** : (int) c : (int) r ..... Get object from table
- (long) **count** : (id) obj ..... Count occurrences of object

- (BOOL) **has** : (id) obj ..... Test presence object
- **each** : (SEL) sel ..... Perform sel for every object

## DTree

### Methods

- **init** ..... Init empty tree
- **shallowFree** ..... Free the tree, not the objects
- **free** ..... Free the tree and the objects
- (long) **length** ..... Return the number of objects in tree
- (BOOL) **isEmpty** ..... Check if the tree is empty
- (BOOL) **has** : (id) obj ..... Test if an object is in the tree
- (long) **count** : (id) obj ..... Count occurrences of object
- **each** : (SEL) sel ..... Perform sel for each object

## DTreeIterator

### Methods

- **init** : (DTree \*) tree  
    └ Init iterator with tree, move to root
- (id) **first** ..... Move iterator to first node
- (id) **next** ..... Move iterator to next node
- (id) **prev** ..... Move iterator to previous node
- (id) **last** ..... Move iterator to last node
- (BOOL) **move** : (id) obj ..... Move iterator to object
- (BOOL) **isFirst** ..... Test for first node
- (BOOL) **isLast** ..... Test for last node
- (BOOL) **isRoot** ..... Test for root node
- (BOOL) **hasChildren** ..... Test for children for this node
- (id) **root** ..... Move to the root node
- (id) **parent** ..... Move to the parent node
- (id) **child** ..... Move to the (first) child node from parent
- (id) **object** : (id) obj ... Set the object in the current node
- (id) **object** ..... Return the object in the current node
- **prepend** : (id) obj ..... Prepend new child for current parent
- **append** : (id) obj ..... Append new child for current parent
- **before** : (id) obj ..... Insert new node before current node
- **after** : (id) obj ..... Insert new node after current node
- (id) **remove** ..... Remove current node, no children

---

Version 0.7.0. This card may be freely distributed under the terms of the GNU general public licence

Copyright © 2003-2005 by Dick van Oudheusden