C19: Collection Classes

(don't forget to look at all the online code examples)

Arrays (just a remainder)

- 3-step process:
  - declare: `static public CardPile[] allPiles;`
  - define: `allPiles = new CardPile[27];` or `int[] primes = {2,3,5,7};`
  - intialize: `for(int i = 0; i< allPiles.length; i++) allPiles[i] = new TablePile(i);`
- Bounds are checked! 0 .. length-1, else throw IndexOutOfBoundsException
- Have a uniform type (primitives or classes), but is polymorphic: e.g. Object[] can hold instances of any class
- Clonable: creates a shallow copy:
  - `int[] numbers = primes.clone();`

Collections (like Vector, ..)

- Maintain their values as type Object ➔
  - Cannot store primitives directly, must use wrapper classes (Integer, Boolean, …)
  - Must cast back retrieved objects to their original type
- Can be seen as a linear sequence of elements: support Enumeration interface

Enumerators

- Uniform way of iterating through the elements of whatever type collection:
  `boolean hasMoreElements()`
  `Object nextElement()`
- Always invoke these two in tandem,
  - Never call nextElement() without first checking hasMoreElements() for truth
  - Never call nextElement() twice in a row
- Typical code patterns are:

```
for(Enumeration e = htab.elements(); e.hasMoreElements(); )
    System.out.println(e.nextElement);
```

  - Or:

```
Enumeration e = htab.elements();
while (e.hasMoreElements()) System.out.println(e.nextElement);
```

Vector

- Similar to arrays, but more general operations supported:
  - (automatically) expandable!

Size: size(), isEmpty(), capacity(), setSize(int)
Access: contains(Object), firstElement(), lastElement(), elementAt(int)
Insert/modify: addElement(Object), setElementAt(Object,int),insertElement(Object,int)
Remove: removeElementAt(int),removeElement(Object), removeAllElements()
Search: indexOf(Object), lastIndexOf(Object)
Misc: clone(), toString()

Use Vector as ..

Array:
v.setElementAt(v.elementAt(37)+12,5);
Stack: addElement(), lastElement(), removeElementAt(v.size()-1)
Queue: add to back, remove from front: addElement(), firstElement(), removeElementAt(0)
Set: contains(), addElement(), removeElement()

Use Vector as a List

- Insert and remove at any location, locate
  insertElementAt(Object,int)
  removeElementAt(Object,int)
- Not a linked list: arbitrary inserts/remove might be expensive, need to shift around vector contents

Stack collection

- Subclass of Vector:
  Object push(Object)
  Object peek()
  Object pop()
  int size()
  boolean empty()
  int search(Object): -1 if not found, 1 for top of stack, … (book is wrong)
  (cf. chapter 10 for pro/cons "Stack extends Vector")

BitSet

- Expandable like Vector, alternative to boolean[], supports more methods (see Sieve example):

BitSet(int)	constructor, all bits are off (0) initially
void set(int)	set a bit
boolean get(int)	get bit status
void clear(int)	clear a bit (set to 0)
void or(BitSet)	compute logical-or of two bitsets
void and(BitSet)	
void xor(BitSet)	
String toString():	nice list of comma-separated on-positions

Abstract class Dictionary

- Relate arbitrary values to keys:
  Object get(Object key)
  Object put(Object key, Object value)
  Object remove(Object key)
- Hashtable: subclass, adds useful methods
  boolean containsKey(Object key)
  boolean containsValue(Object value)
  Enumeration elements()
  Enumeration keys()
  void clear()
  (see Concordance code example)

Properties

- Subclass of HashTable managing String key/value pairs including storage on disk:
  void load(InputStream)
  void save(OutputStream o, String header)
  – see Properties code example

Order

- Initially Java had no ordered collections, but since 1.2 is has, using a mechanism similar to the one in described in section 19.8
- Look for: Comparator, Comparable, static sort method in class Arrays, SortedMap, …