* + Ex: SongFilter
    - an object that represents a search criteria to be applied to a collection of songs
  + How to address situation when an outside user must apply something to a collection
    - one way is to use public Song[] getSongs()
    - but, the collection may be large so we don't want to provide the whole thing at once
      * this creates an array to hold the songs
    - strategy 2: provide index access to colleciton
    - strategy 3: internalize a cursor
      * Ex: public void resetSongCursor(), getNextSong(), isCursorAtEnd()
      * drawbacks: can't have two traversals going at the same time
  + iterator design pattern
    - create a whole new object called the 'iterator' that encapsulates the state of the item traversal
    - so if more than 1 items need to access the elements of a collection, it can ask the iterator object to provide this info
    - the iterator undertands the details of the underlying collection
      * manages the order of items
        + may want a traversal that is not just first to last, or if the underlying collection may not have a natural linear order
      * manages state of traversal
        + allows traversal to be picked up again later
    - two components:
      * collection object - the "iterable"
        + must have a method that returns an object that acts as an iterator
      * the iterator
        + provides access to elements in turn

at the very least the iterator:

tests whether more items exist

retrieving the next item

other possible features:

methods that remove an item safely

to peek at the next item

to reset the traversal

* + - Java Iterator Pattern Interfaces
      * java provides two generic interfaces for supporting the iterable design pattern
        + implemented by various collection types such as List<E>, Map<E>, Set<E>, etc.

all of these collection types are of type Iterable<E>

the Iterator<E> has the following method:

boolean hasNext(), E next(), void remove()