(b) Iterating IntBags

~~The Java API includes an interface called Iterator that is designed to retrieve each of the objects from a set of objects, once and once only. This allows programs to process each item in the set without being aware of the underlying implementation (which means it can be changed at will without affecting the client program.) Your task is to provide this interface for your IntBag class (from Lab 1.)~~ Note: The IntBag class was designed to store primitive int types, rather than Java Object types. Since the Iterator next() method must return an Object, you will need to return the corresponding int in an Integer wrapper. Clearly, this is not an entirely satisfactory solution and you might consider extending Iterator to produce IntIterator, which provides an additional int nextInt() method.  
  
There are several ways you might think of coding this problem. Clearly, you must have a value (say index) that keeps track of the position of the next item to be retrieved from the IntBag collection. This must be initialized to zero when an iteration begins, and be incremented each time a call is made to get the next element (such call returning the element at the index location within the set.) The hasNext() method simply determines whether the index value is less than the number of elements in the set or not. Putting this code inside the IntBag class is problematic; how can index be reset to zero so that the elements can be iterated through several times, and how can multiple independent iterations through the set be managed?  
  
~~A simple solution to this is to write a new class, say, IntBagIterator, which implements Iterator. This class needs to have only two properties; aBag (a reference to the IntBag it is to iterate through) and index (the position of the next element to return from aBag.) Code your IntBagIterator as an inner class of your IntBag. Finally, add a method Iterator iterator() to your IntBag class.~~ ~~This should create an instance of IntBagIterator and return it as the interface type. Test your design using a class, TestIterators, which includes the following code:~~

Bag bag = new Bag();

// … insert some elements

Iterator i, j;

i = bag.iterator();

while ( i.hasNext() )

{

System.out.println( i.next() );

j = bag.iterator();

while ( j.hasNext() )

{

System.out.println( "--" + j.next() );

}

}

~~Notice that i & j are of the interface type, Iterator. This is valid since IntBagIterator implements Iterator and so is an Iterator.~~

Now implement a different kind of iterator class named IntBagStepIterator. This class should take a constructor parameter that denotes a step value m. Add a different constructor to your intBag class which takes a step value too and creates IntBagStepIterator with this stepping value. This object must be returned with iterator() function of your intBag class. Using the m value next() and hasNext() functions in IntBagStepIterator must look for the current element + m th element in the intBag. For example, assume that intBag contains the following set of int values:  
  
1,2,3,4,5,6,7, …, 20  
  
An iterator with step value m = 2 should return the following set of elements when next() function is called up to the end of the int array:  
  
1,3,5,7,9,11,13,15,17,19

*Note: can you make your IntBag implement Java's Iterable interface now? Java's ArrayList class has a similar method that allows you to iterate through its elements. Interestingly, the Java API implements the same design pattern that corresponds to our IntBagIter*