An iterator over the even values of an array

Here is the start of a class whose objects are iterators over even values of Integer arrays. It implements Iterator<Integer>. Because of that, we stub in required methods hasNext() and next(). The first statement of next() throws the necessary exception if hasNext() is false. Method next() should always start with such a call on hasNext().

```
/** An instance is an iterator over the even values of an array. */
public class EvenIterator implements Iterator<Integer> {

   /** = there is another element to enumerate. */
   public @Override boolean hasNext() {
      return ?;
   }

   /** Return the next element to enumerate.
     * Throw a NoSuchElementException if there is no next element. */
   public @Override Integer next() {
      if (!hasNext()) throw new NoSuchElementException();
      …
      return ?;
}
```

The next step is to write the constructor. It should have as parameter the array whose even values are to be enumerated. We declare a field to contain the array and start the constructor (leaving the other methods out):

```
/** An instance is an iterator over the even values of an array. */
public class EvenIterator implements Iterator<Integer> {
   private Integer[] b; // array whose even values are to be enumerated.

   /** Constructor: An iterator over the even values of b. */
   public EvenIterator(Integer[] b) {
      this.b= b;
   }
}
```

Now, this class must keep track of the elements of b as they are being enumerated. For this purpose, we introduce a second field, which contains the index of the *next* element to be enumerated. It equals b.length if all values have been enumerated:

```
private int n;      // b[n] is the next item to enumerate (n = b.length
                    // if there are no more to enumerate). */
```

With that field, it is easy to complete function hasNext():

```
/** = there is another element to enumerate. */
public @Override boolean hasNext() {
      return n < b.length;
}
```

The constructor must initialize field n to the index of the first even element. Also, method next() will have to change n when it returns b[n]. Since n may have to be changed in two places, we write a method for it. We suggest that you stop the video at this point and study the method specification and body.

```
/* Increase n to the index of the next element in b that
 * is even ---or to b.length if there no such elements.
 * Precondition: n < b.length. */
private void fixNext() {
      n= n+1;
      while (n < b.length && b[n]%2 == 1) n= n+1;
}
```

We can now change the constructor to truthify the definition of n, setting n to -1 first so that the call on nextElement will start looking at b[0]. Similarly we write the body of method getNext(): save the return value in r, call fixNext to increment n appropriately, and return r.

An iterator over the even values of an array

Summary

Most iterators have this form.

First, there are fields to represent what is being enumerated –an array like b, the head of a linked list, something that describes a set.

Second, there are fields, like n, to help describe what remains to be enumerated.

Depending on how complicated it is to find the next element to be enumerated, there may be a method like fixNext().

The constructor saves its parameters and truthifies the class invariant.

Method hasNext() usually has a simple body.

Finally, method next() always makes sure there is an element to enumerate. It usually saves the return value, fixes the fields to describe the *next* element to be returned, and returns the saved value.

```
/** An instance is an iterator over the even values of an array. */
public class EvenIterator implements Iterator<Integer> {

   private Integer[] b; // The array whose even values are to be enumerated
   private int n;       // b[n] is the next item to enumerate (= b.length if
                        // there are no more to enumerate). */

   /** An iterator over the even values of b. */
   public EvenIterator(Integer[] b) {
      this.b= b;
      n= -1;
      fixNext ();
   }

   /** = there is another element to enumerate. */
   public @Override boolean hasNext() {
      return n < b.length;
   }

   /** Return the next element to enumerate.
    * Throw a NoSuchElementException if there is no next element. */
   public @Override Integer next() {
      if (!hasNext()) throw new NoSuchElementException();
      Integer r= b[n];
      fixNext();
      return r;
   }

   /* Increase n to the index of the next element in b that
    * is even ---or to b.length if there no such elements.
    * Precondition: n < b.length */
   private void fixNext() {
      n= n+1;
      while (n < b.length && b[n]%2 == 1) n= n+1;
   }
}
```