Assignment	1. Write a class named ArrayIterator that implements the Iterator interface and iterates over elements in an array, but skips null elements in the array. 2. For type safety, include a type parameter <T> in the class. 3. Use the package **`ku.util`** for your code.
What to Submit	1. Create an empty repository on Github Classroom using this link: **`https://classroom.github.com/a/g0EhuapO`** 2. Clone the repository. Add your own README.md and .gitignore. 3. Submit your source code in the "`src/ku/util`" directory. Push to Github.
Testing	You should test and review your own code. There are some JUnit tests in week4/ArrayIteratorTest.java, but they do not test everything!

Iterators

Many collections and data structures provide an *Iterator* so we can iterate over all the elements in the collection *without knowing the structure* of the collection.

In Java, an *Iterator* is any object that implements the java.util.Iterator interface. This interface has a type parameter that describes the type of element the Iterator returns.

Iterator Interface in Java

The **`java.util.Iterator`** interface has 3 methods. The interface has a type parameter (usually shown as "T" or "E"). If you omit the type parameter, the default value is **`Object`**. Here are the 3 methods (shown with and without type param):

Type parameter T	No type parameter	Meaning
`T next( )`	**`Object next( )`**	Return the *next non-null* element in the array. If there are no more elements, it throws NoSuchElementException.
`boolean hasNext()`	**`boolean hasNext()`**	Returns true if **`next()`** can return another non-null array element, false if no more elements.
`void remove()`	**`void remove( )`**	(Optional) Remove most recent element returned by next() from the array by setting it to **`null`**. This method may only be called once after a call to next(). If this method is called without calling next(). or called more than once after calling next(), it throws IllegalStateException.

Example:

Scanner is a String Iterator. In the Java API doc for Scanner, it shows that Scanner implements Iterator<String>. This means that next( ) will return a String. For example:

```
    Scanner input = new Scanner( "Iterating is so easy!" );
    while( input.hasNext() ) {
        String s = input.next();
        System.out.println( s );
    }
```

```
Iterating
is
```

so
easy!

Assignment

Arrays don't have an Iterator, but it would be really useful to have one. Write an ArrayIterator class in the package **ku.util** that provides an *Iterator* for any array.

For *convenience*, we will design the Arrayiterator so it will skip null elements in the array.

1. Write a class named ArrayIterator that implements java.util.Iterator.

2. Use a *type parameter* in the class declaration and methods. Declare the class like this:

```
public class ArrayIterator<T> implements Iterator<T>
```

T is a *type parameter*, which is a placeholder for the name of a class or Interface. We will study type parameters later, but you can use it by just following the sample code below.

3. The *type parameter* should match the type of elements in the array. If we have an array of String, we want ArrayIterator to return Strings so we would write "new ArrayIterator<String>(array)". If we have an array of Student, we would write "new ArrayIterator<Student>(students)" and "T" would become Student.

Define ArrayIterator like this. The "T" means the type of thing in the ArrayIterator. At run-time it will be replaced by the name of an actual class (like Student if we ask for ArrayIterator<Student>).

```
package ku.util;

//TODO write good Javadoc
public class ArrayIterator<T> implements Iterator<T>  {
    /** attribute for the array we want to iterate over */
    private T[ ] array;

    /**
     * Initialize a new array iterator with the array to process.
     * @param array is the array to iterate over
     */
    public ArrayIterator(T[] array) {
        this.array = array;
        //TODO: initialize any other variables you need
    }

    /**
     * Return the next non-null element from array, if any.
     * @return the next non-null element in the array.
     * @throws NoSuchElementException if no more elements to return
     */
    public T next( ) {

    }
    //TODO the hasNext() and remove() methods don't use the type
    // parameter, so you should have no problem writing them.
}
```

4. The *constructor* has a parameter that is an array of type T. In Java, you can use a type parameter just like a class name (except that you can't create "new" objects using a type parameter).

5. ArrayIterator may *not* use any Java collections (like ArrayList). ArrayIterator needs only a reference to the **`array`** and a variable (or two) to keep track of the next element to return..

6. The next( ) and hasNext( ) methods should *skip* **`null`** values (see example below).

7. If the user calls next( ) when there are no more elements, next throws a NoSuchElementException. Here is how to throw an exception:

```
throw new NoSuchElementException( );
```

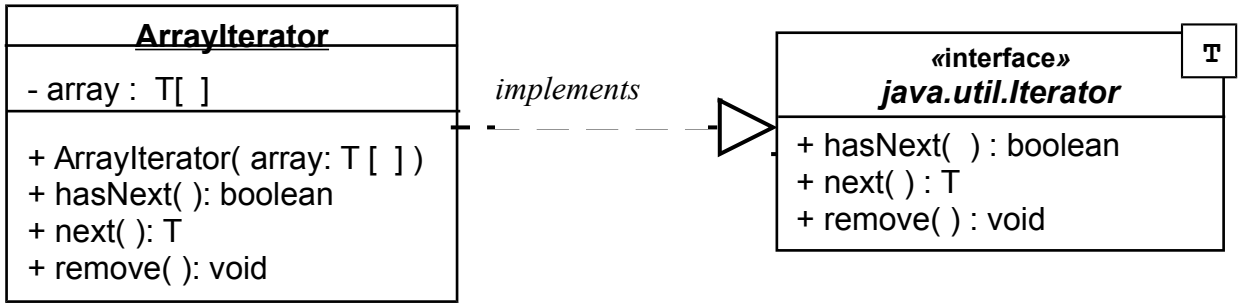
The code immediately exits the current method when you throw exception, so there is no "return" after you throw an exception. See the Programming Notes below for more on throwing an exception.

remove( ) method

This method is optional. You can leave the remove() method empty.

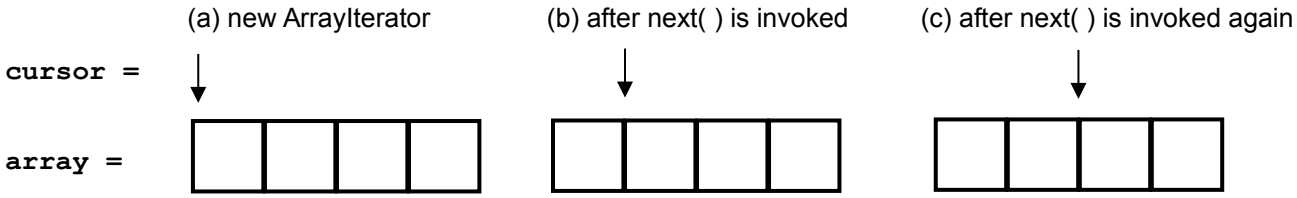
Extra credit will be given if you write a 100% correct `remove` method (see specification above).

Class Diagram for ArrayIterator



Programming Notes

1. An Iterator needs a variable (often called the **cursor**) to remember its position in the collection. Initially the cursor points to the first element. Each time **next** is called, the Iterator returns the current element and increments the cursor. hasNext() may also advance the cursor to skip null elements.



2. *The* **`hasNext`** *method* ***does most of the work!*** It is the job of hasNext to decide if there is another element available and *move the cursor* to the location of the next non-null element.

3. **Don't duplicate** code or logic! The next method should ask hasNext if there is another element, and let hasNext do the work of moving the cursor. Don't copy the hasNext logic into the next method.

4. It is legal for the user to call hasNext( ) *many times* consecutively without calling next. The iterator must not skip any elements if the user does this!

```
iterator.hasNext();
iterator.hasNext();  // no change. Duplicate calls to hasNext don't change the iterator.
iterator.hasNext();  // no change, again.
```

5. It is also legal for the user to call next *without* calling hasNext. Therefore, you must not *assume* the user will always call hasNext before next.

```
String [] array = { "apple", "banana", null, "carrot" };
ArrayIterator<String> iter = new ArrayIterator( array );
iter.next( );     // returns "apple"  User is not required to call hasNext.
iter.hasNext( ); // true
iter.hasNext( ); // true again
iter.hasNext( ); // true again        User can call hasNext many times
iter.next( );     // returns "banana"
iter.next( ):     // returns "carrot" (skip over null element)
iter.hasNext();  // false
iter.next( );     // throws NoSuchElementException
```

6. To throw an Exception, simply write `throw new NoSuchElementException( )`. Throwing an exception causes an immediate return from the method. Don't write return after throw. For example:

```
/** get the n-th element from double array[ ] */
public double get(int n) {
    if (n >= 0 && n < array.length) return array[n];
    else throw new NoSuchElementException( );
     // "throws" exits from the method so don't write "return" here.
}
```

Example using BlueJ Interactive Mode

```
> String [] fruit = { "apple", null, null, "banana"};
> ArrayIterator<String> it = new ArrayIterator<String>(fruit);
> it.hasNext()
true
> it.next()
"apple"
> it.next()
"banana"
> it.hasNext()
false          // no more elements: hasNext() is false forever
> it.next( )
java.util.NoSuchElementException at ArrayIterator:xx
```

Example using an empty array:

```
> Object [ ] array = new Object[1];  // array containing null
> ArrayIterator it = new ArrayIterator( array );
> it.hasNext( )
false
> it.next()
java.util.NoSuchElementException at ArrayIterator:xx
```