

Assignment	<ol style="list-style-type: none"> 1. Write a class named <code>ArrayIterator</code> that implements the <code>Iterator</code> interface and iterates over elements in an array, but skips <code>null</code> elements in the array. 2. Include a type parameter <code><T></code> in the class. 3. Use the package <code>ku.util</code> for your code.
What to Submit	<ol style="list-style-type: none"> 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 "ku/util" directory. Push to Github.
Individual Work	This is an individual assignment. Do not ask other students for help or share code. If you need help, please ask the TAs or instructor.

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 specifies 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 parameter):

Type parameter T	No type parameter	Meaning
<code>T next()</code>	<code>Object next()</code>	Return the <i>next non-null</i> element in the array. If there are no more elements, it throws <code>NoSuchElementException</code> .
<code>boolean hasNext()</code>	<code>boolean hasNext()</code>	Returns <code>true</code> if <code>next()</code> can return another non-null array element, <code>false</code> if no more elements.
<code>void remove()</code>	<code>void remove()</code>	(Optional) Not used in <code>ArrayIterator</code>

Example:

`Scanner` is a `String` *Iterator*. The Java API doc for `Scanner` states that `Scanner` implements `Iterator<String>`. This means that `next()` will return a `String`. For example:

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

```
Iterating
is
easy!
```

Assignment

Arrays don't have an *Iterator*, but it would be 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 skips `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 `next` method. 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.

3. When we create an `ArrayIterator` object, the value of the type parameter should match the type of values in the array. For an array of `Student` objects we would write:
`"new ArrayIterator<Student>(students)"`.

Define `ArrayIterator` like this. The "`T`" means the type of object that the `next()` method returns.

```
package ku.util;

//TODO write good Javadoc

public class ArrayIterator<T> implements Iterator<T> {
    private T[] array;
    //TODO: define any other variables you need (a cursor)
    /**
     * 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;
    }

    /**
     * @see java.util.Iterator#next()
     */
    @Override
    public T next() {

    }

    @Override
    public boolean hasNext() ...
}
```

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 cannot create "new" objects using a type parameter, e.g. "new T()").
5. `ArrayIterator` may not use any Java collections (like `ArrayList`). `ArrayIterator` needs only a reference to the **array** and a variable to keep track of the next element it should return..
6. The `next()` and `hasNext()` methods should *skip null* values (see example below).
7. Do not expect the user to always call `hasNext()` before `next()`. He may not call `hasNext()` at all, or call it many times consecutively.
8. If the user calls `next()` when there are no more elements, `next` throws a `NoSuchElementException`. Here is how to throw an exception:

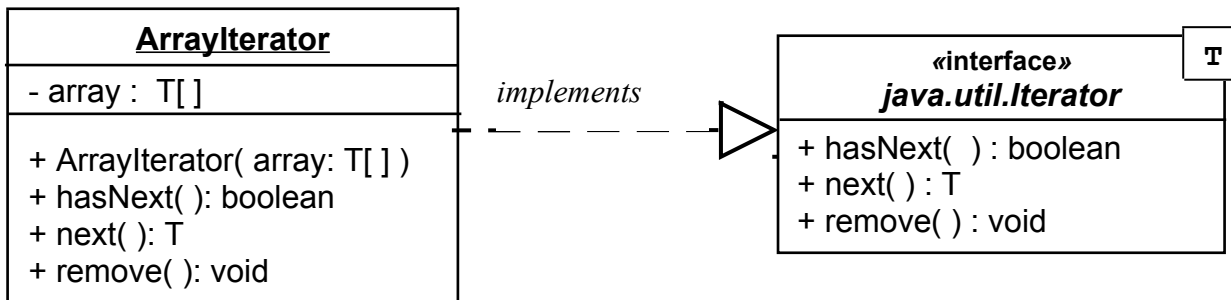
```
throw new NoSuchElementException("No more elements");
```

The code immediately exits when you throw an exception, so there is no "return" after you throw an exception.

remove() method

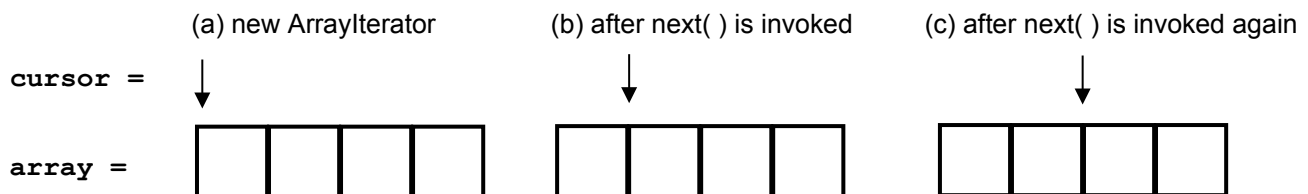
This method is optional. You can leave the remove() method empty or omit it.

Class Diagram for ArrayIterator



Programming Notes

1. An Iterator needs a variable (often called the **cursor**) to remember its position. Initially the cursor points to the first element. **hasNext()** checks if the cursor points to a valid element. If it is not valid, **hasNext** should advance the cursor to a valid element. Each time **next** is called, the Iterator returns the current element and increments the cursor by 1.



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 write duplicate 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 do not 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 calling **next**.

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

```

if (no more elements) {
    throw new NoSuchElementException("No more elements");
}
  
```

Example using BlueJ Interactive Mode

```

> String [] array = { "apple", "banana", null, "carrot", null };
> ArrayIterator<String> iter = new ArrayIterator( array );
> iter.next( )      // User not required to call hasNext() before next()
  
```

```

"apple"
> iter.hasNext( )
true
> iter.hasNext( )      // User can call hasNext many times
true
> iter.next( )
"banana"
> iter.next( )
"carrot"                // iterator skips the null element in array
> iter.hasNext()
false                   // iterator skips the null element in array
iter.next( )            // no more elements, so an exception is thrown
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

```

Testing

You should test your code!

Test at least 3 kinds of situations:

Test Case	Array used in new ArrayIterator() to test next() and hasNext()
Edge Case	1. Empty array String[] array = { }; 2. Array of size 1 containing either data or null. String[] array = { "only one" }; String[] array2 = { null };
Typical Case	1. Array with all data. Number[] array = { new Integer(1), new Double(2), new Long(3) }; 2. Array with some nulls at start or end. String[] array = { null, "second", "third" }; String[] array2 = { "first", "second", "third", null, null, null }; 3. Array with data and nulls mixed together String[] array = { "first", null, "second", null, null, "last" };
Extreme Case	1. Array with all null. String[] empty = new String[100]; // hasNext() should return false 1. Big array with one non-null element somewhere. String[] array = new String[100]; array[1] = "first"; array[50] = "middle";