Assignment	1. Write a class named ArrayIterator that implements the Iterator interface and iterates over elements in an array, but skips null elements. 2. For type safety, include a type parameter <T> in the class. 3. Use the package **ku.util** for your code.
What to Submit	Commit your project to Bitbucket as project name PA2. Both problem 1 (ArrayIterator) and problem 2 (Stack) are part of the same project. Share the project with the TAs.

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.

Examples:

1. Scanner is a String Iterator:

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

2. A List has an iterator() method creates an Iterator for *any* list.

```
List<Coin> list = new ArrayList<Coin>( );
list.add( new Coin(5) );
list.add( new Coin(10) );
//...  add more coins
Iterator<Coin> iterator = list.iterator( );  // create iterator
while( iterator.hasNext() ) {
    Coin coin = iterator.next( );
    System.out.println( coin );
}
```

Assignment

Arrays don't have an Iterator, but it would be really useful to have one. Your assignment is to write an ArrayIterator class 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 above.

3. The *constructor* has one parameter: 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). You can declare a parameter or an attribute using the type parameter:

```
private T[ ] array;
```

4. ArrayIterator may *not* use any Java collections (like ArrayList). ArrayIterator needs only a reference to the array and an index variable (a "cursor") to keep track of the next element to return.

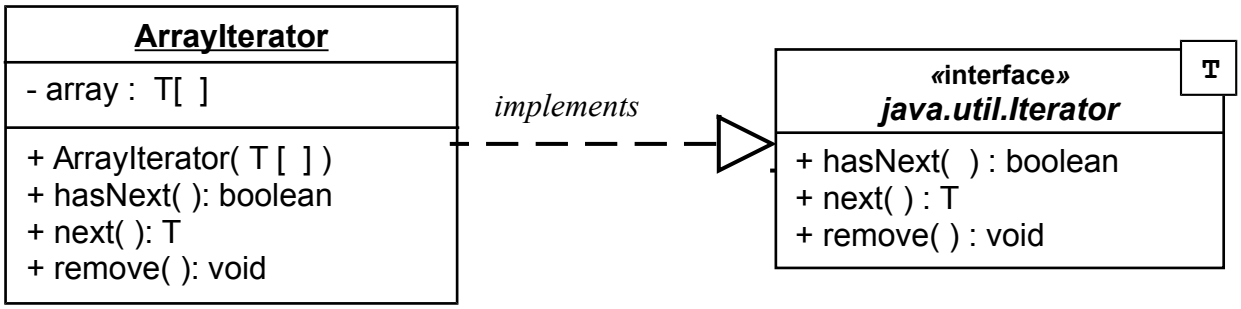
5. The next( ) and hasNext( ) methods should *skip null values* (see example below).

6. The Javadoc for the *Iterator* interface says that if the user calls next( ) when there are no more elements, next throws a NoSuchElementException. Your ArrayIterator should do this, too. To throw an exception, write: throw new NoSuchElementException( );

Methods:

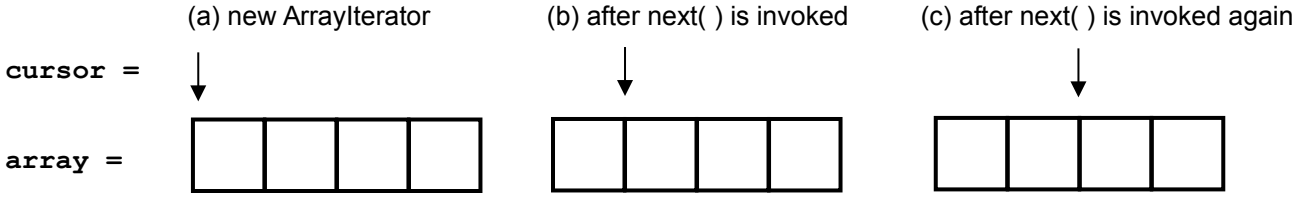
`T next( )`	Return the *next non-null* element in the array. If there are no more elements, throws NoSuchElementException.
`boolean hasNext()`	Returns true if **`next()`** can return another non-null array element, false if no more elements.
`void remove()`	Remove most recent element returned by next() from the array by setting it to null.

Class Diagram for ArrayIterator



Programming Notes

1. An Iterator needs a variable (called the **cursor**) to remember its position in the collection. Initially the cursor points to the first element. Each time **next** is invoked, 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 start 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 skipping nulls. 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();
```

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.

Example using BlueJ Interactive Mode

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

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
```