Problem 284: Peeking Iterator

Problem Information

Difficulty: Medium
Acceptance Rate: 60.90%
Paid Only: No
Tags: Array, Design, Iterator

Problem Description

Design an iterator that supports the `peek` operation on an existing iterator in addition to the `hasNext` and the `next` operations.

Implement the `PeekingIterator` class:

* `PeekingIterator(Iterator<int> nums)` Initializes the object with the given integer iterator `iterator`. * `int next()` Returns the next element in the array and moves the pointer to the next element. * `boolean hasNext()` Returns `true` if there are still elements in the array. * `int peek()` Returns the next element in the array **without** moving the pointer.

Note: Each language may have a different implementation of the constructor and `Iterator`, but they all support the `int next()` and `boolean hasNext()` functions.

Example 1:

Input ["PeekingIterator", "next", "peek", "next", "next", "hasNext"] [[[1, 2, 3]], [], [], [], [], []] **Output** [null, 1, 2, 2, 3, false] **Explanation** PeekingIterator peekingIterator = new PeekingIterator([1, 2, 3]); // [_**1**_ ,2,3] peekingIterator.next(); // return 1, the pointer moves to the next element [1,_**2**_ ,3]. peekingIterator.peek(); // return 2, the pointer does not move [1,_**2**_ ,3]. peekingIterator.next(); // return 2, the pointer moves to the next element [1,2,_**3**_] peekingIterator.next(); // return 3, the pointer moves to the next element [1,2,3] peekingIterator.hasNext(); // return False

Constraints:

* `1 <= nums.length <= 1000` * `1 <= nums[i] <= 1000` * All the calls to `next` and `peek` are valid. * At most `1000` calls will be made to `next`, `hasNext`, and `peek`.

Follow up: How would you extend your design to be generic and work with all types, not just integer?

Code Snippets

C++:

```
/*
* Below is the interface for Iterator, which is already defined for you.
* **DO NOT** modify the interface for Iterator.
*
* class Iterator {
* struct Data;
* Data* data;
* public:
* Iterator(const vector<int>& nums);
* Iterator(const Iterator& iter);
*
* // Returns the next element in the iteration.
* int next();
*
* // Returns true if the iteration has more elements.
* bool hasNext() const;
* };
*/

class PeekingIterator : public Iterator {
public:
PeekingIterator(const vector<int>& nums) : Iterator(nums) {
// Initialize any member here.
// **DO NOT** save a copy of nums and manipulate it directly.
// You should only use the Iterator interface methods.

}

// Returns the next element in the iteration without advancing the iterator.
int peek() {
}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
int next() {

}

bool hasNext() const {

}
};
```

Java:

```
// Java Iterator interface reference:
// https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

class PeekingIterator implements Iterator<Integer> {
public PeekingIterator(Iterator<Integer> iterator) {
// initialize any member here.

}

// Returns the next element in the iteration without advancing the iterator.
public Integer peek() {

}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
@Override
public Integer next() {

}

@Override
public boolean hasNext() {

}
}
```

Python3:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator:
# def __init__(self, nums):
# """
# Initializes an iterator object to the beginning of a list.
# :type nums: List[int]
# """
#
# def hasNext(self):
# """
# Returns true if the iteration has more elements.
# :rtype: bool
# """
#
# def next(self):
# """
# Returns the next element in the iteration.
# :rtype: int
# """

class PeekingIterator:
def __init__(self, iterator):
"""
Initialize your data structure here.
:type iterator: Iterator
"""



def peek(self):
"""
Returns the next element in the iteration without advancing the iterator.
:rtype: int
"""



def next(self):
"""
:rtype: int
"""
```

```
def hasNext(self):
"""
:rtype: bool
"""


# Your PeekingIterator object will be instantiated and called as such:
# iter = PeekingIterator(Iterator(nums))
# while iter.hasNext():
# val = iter.peek() # Get the next element but not advance the iterator.
# iter.next() # Should return the same value as [val].
```