Peeking Iterator

Given an Iterator class interface with methods: `next()` and `hasNext()`, design and implement a PeekingIterator that support the `peek()` operation -- it essentially peek() at the element that will be returned by the next call to next().

Here is an example. Assume that the iterator is initialized to the beginning of the list: `[1, 2, 3]`.

Call `next()` gets you 1, the first element in the list.

Now you call `peek()` and it returns 2, the next element. Calling `next()` after that ***still*** return 2.

You call `next()` the final time and it returns 3, the last element. Calling `hasNext()` after that should return false.

1. Think of "looking ahead". You want to cache the next element.
2. Is one variable sufficient? Why or why not?
3. Test your design with call order of `peek()` before `next()` vs `next()` before `peek()`.
4. For a clean implementation, check out Google's guava library source code.

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

Credits:
Special thanks to @porker2008 for adding this problem and creating all test cases.

Solution 1

```java
class PeekingIterator implements Iterator<Integer> {
    private Integer next = null;
    private Iterator<Integer> iter;

    public PeekingIterator(Iterator<Integer> iterator) {
        // initialize any member here.
        iter = iterator;
        if (iter.hasNext())
            next = iter.next();
    }

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

    // hasNext() and next() should behave the same as in the Iterator interface.
    // Override them if needed.
    @Override
    public Integer next() {
        Integer res = next;
        next = iter.hasNext() ? iter.next() : null;
        return res;
    }

    @Override
    public boolean hasNext() {
        return next != null;
    }
}
```

cache the next element. If next is null, there is no more elements in iterator.

written by chouclee original link here

Solution 2

Since `Iterator` has a copy constructor, we can just use it:

```
class PeekingIterator : public Iterator
{
public:
    PeekingIterator(const vector<int> &nums) : Iterator(nums)
    {
    }

    int peek()
    {
        return Iterator(*this).next();
    }

    int next()
    {
        return Iterator::next();
    }

    bool hasNext() const
    {
        return Iterator::hasNext();
    }
};
```

written by efanzh original link here

Solution 3

```
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() {
    if(hasNext()){
        Iterator it(*this);
        return it.next();
    }
}

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

bool hasNext() const {
    Iterator::hasNext();
}
```

};

written by leebear original link here