Hyperion Development

Task: Iterators

www.hyperiondev.com

Introduction

Collecting information

In this task you are going to learn about an iterator which is used for looping through various collections such as HashMaps, LinkedLists, ArrayLists, etc. You should recall from previous tasks in which we dealt with collections that we always used some form of a basic loop to iterate through the specified collection. This basic loop was simplistic, however offers very little functionality in way which data could be extracted or identified in the respective collections. Thus the iterator is introduced, although it features a basic design which is slightly more complicated than a basic loop, it boasts a vast spectrum of methods which can be used to manipulate or customise how data is extracted from collections. Let's get started!

-The Hyperion Team

Identifying and Defining Iterators

An Iterator is an object that enables to cycle through a collection, obtain or remove elements. Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator() method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.

The Iterator class provides the following methods:

- **hasNext():** Returns true if there is at least one more element; otherwise, it returns false.
- **next():** Returns the next object and advances the iterator.
- **remove():** Removes the last object that was returned by next from the collection.

The Iterator class must be imported from the java.util package.
Example of Iterator implementation:

```
import java.util.Iterator;
import java.util.LinkedList;

public class MyClass {
  public static void main(String[ ] args) {
 LinkedList<String> animals = new LinkedList<String>();
 animals.add("fox");
 animals.add("cat");
 animals.add("dog");
 animals.add("rabbit");

 Iterator<String> it = animals.iterator();
 String value = it.next();
 System.out.println(value);
}
}
//Outputs "fox"
```

Note: *it.next() returns the first element in the list and then moves the iterator on to the next element. Each time you call it.next(), the iterator moves to the next element of the list.*

Properties of ListIterators

In the above section, we discussed Iterators in Java using which can traverse a List or Set in a forward direction. Here we will discuss the ListIterator which allows us to traverse the list in both directions (forward and backward).

The ListIterator class provides the following methods:

- **add(Object obj):** Inserts obj into the list in front of the element that will be returned by the next call to next().
- **hasNext():** Returns true if there is a next element. Otherwise, returns false.
- **hasPrevious():** Returns true if there is a previous element. Otherwise, returns false.
- **next():** Returns the next element. A NoSuchElementException is thrown if there is not a next element.
- **nextIndex():** Returns the index of the next element. If there is not a next element, returns the size of the list.

- **previous():** Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.

- **previousIndex():** Returns the index of the previous element. If there is not a previous element, returns -1.

- **remove():** Removes the current element from the list. An IllegalStateException is thrown if remove() is called before next() or previous() is invoked.

- **set(Object obj):** Assigns obj to the current element. This is the element last returned by a call to either next() or previous().

Note: *The ListIterator class must be imported from the java.util package.*

Instructions

First read example.java, open it using jGRASP (Right click the file and select 'Open with jGRASP").

- In this folder there is a file called example.java

- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the 'Open with' menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.

- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.

- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task 1

Follow these steps:

- Create a new java file called BasicIterator.java
- Refer to the ArrayList you created in the ***Lists*** Task.
- Define an iterator loop to do the following:
  - Check if the next object exists in the ArrayList
  - Print out the next object
  - Remove the object that was returned
- Define a count variable to count how many items were in your ArrayList based of how many time the check for the next object returns true.
- Print out the count variable
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Create a new java file called BasicListIterator.java
- Refer to the ArrayList you created in the ***Lists*** Task.
- Add three objects to the list.
- Utilise the nextIndex() method until the size of the list is returned.
- Iterate through the loop, printing both the previous and next object for each index.
- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 19/02/2016.
Authors: Jared Ping.
Main trainer: Umar Randeree.
Task Feedback link: Hyperion Development Feedback.