# Performance evaluation of list iteration methods in Java: an empirical study

**Evaluación del rendimiento de los métodos de iteración de listas en Java: un estudio empírico**

**Author:**

**Saman A. Barakat**[1]
**Qusay I. Sarhan**[2]

**SCIENTIFIC RESEARCH**

**How to cite this paper:**
Baraket, S.A. and Sarhan, Q.I., Performance evaluation of list iteration methods in Java: an empirical study, Kurdistan, Irak. Innovaciencia. 2018; 6(1): 1-6.
http://dx.doi.org/10.15649/2346075X.467

**Reception date:**
Received: 16 August 2018
Accepted: 25 November 2018
Published: 28 December 2018.

**Keywords:**
Performance evaluation; Lists; List iteration methods; List iteration time; Test methodology.

## ABSTRACT

**Introduction:** Lists are used in various software applications including web applications, desktop applications, and Internet of Things (IoT) applications to store different types of items (e.g. country name, product model, and device category). Users can select one or more of these items to perform specific tasks such as filling forms, ordering products, reading device data, etc. In some software applications, lists store a huge number of items to be iterated over in order to know what users have selected. From a software development perspective, there are a number of methods to iterate over list items. **Materials and Methods**: In this paper, five list iteration methods: Classic For, Enhanced For, Iterator, List Iterator, and For Each have been compared experimentally with each other with regard to their performance (execution time required to iterate over list items). Thus, a number of experimental test scenarios have been conducted to obtain the comparison results. **Results and Discussion**: The experimental results of this study have been presented in Table 4. **Conclusions**: Overall performance evaluation showed that Iterator and List Iterator methods outperformed other list iteration methods in all test scenarios. However, List Iterator outperformed Iterator when the list size was small. On the other hand, Iterator outperformed List Iterator when the list size was large.

[1] Software Engineering and Embedded Systems (SEES) Research Group, Department of Computer Science, College of Science, University of Duhok, Duhok, Kurdistan Region, Iraq, Email: saman.barakat@uod.ac.

[2] Software Engineering and Embedded Systems (SEES) Research Group, Department of Computer Science, College of Science, University of Duhok, Duhok, Kurdistan Region, Iraq, Email: qusay.sarhan@uod.ac.

## INTRODUCTION

In the last few years, the demand of developing various software applications has been increased exponentially. Software developers use different types of programming languages to build software applications that cover many aspects of human day-to-day activities. Java programming language is one of the dominant languages in software industry nowadays. It is used to develop many types of software applications such as web applications, desktop applications, IoT applications, mobile applications, etc. [1-3]. Some Java based software applications utilize collections data structures (e.g. lists, vectors, or queues) to store/retrieve information required to achieve the goals they built for and to satisfy user needs [4]. For instance, lists can be used to store (a) Names of countries to be used in a form to allow users to select their countries. (b) Languages to be used in a form to allow users to select the languages they speak. (c) Names of products to allow users to select their preferable products to buy. In order to know what users have selected from a list, all list items have to be iterated over. In this context, Java provides developers with five methods [5] : Classic For, Enhanced For, Iterator, List Iterator, and For Each to iterate over list items. In this paper, these methods have been compared experimentally with each other to evaluate their performance in terms of time required to iterate over list items.

## RELATED WORKS

This section presents briefly the most relevant studies and works to the scope of this paper. In literature, many studies have compared the performance of Java with other programming languages. For example, Togashi and Klyuev [6] have evaluated the performance of concurrency (multithreading) and compile time of both Java and Go languages using simple matrix multiplication. Also, the authors in [7], have compared Java and Ruby languages in respect to multithreading. They have used some sort algorithms and simple matrix multiplication in the evaluation process. Hundt [8] has evaluated the performance of loops in Java, Go, C++, and Scala languages in terms of execution time. Many other studies have compared and evaluated some aspects of java itself without comparing with other programming languages. For example, the authors of [9, 10] have compared and evaluated native java arrays with arrays from external java libraries in terms of storage capabilities and execution time. Gupta and Ashraf [4] have developed a Java based collection framework that uses lists and many other item/element storage techniques. And, they have compared it with other ready-made collection framework in terms of storage attributes, degree of simplicity, etc. However, the study have not evaluated the proposed collection framework from a performance (e.g. execution time required for iterating list items using different methods) point of view. It is worth mentioning that all the studies included in this section were useful in providing outstanding explanation of java performance evaluation alongside many directions. Besides, they were valuable in providing a general evaluation metrics for this study. To the best of our knowledge, no previous comprehensive study in the literature compared practically between Java list iteration methods in regard to their execution time. Thus, this was the rationale behind this study to be conducted.

## TEST METHODOLOGY

This paper uses a test methodology adapted from Sarhan and Gawdan [11] research paper. The methodology represents the conditions, scenarios, metrics, and testbed setup that have been applied to compare experimentally the performance of the used list iteration methods in terms of list iteration time.

### 1. Test Conditions

The following test conditions have been considered in this study:

- Every test scenario has been applied on each list iteration method with the same scenario related parameters: list size and item size. In this respect, list size is the number of items stored in a list. Whereas, item size is the size (in bytes) of each item in a list.
- The test application used in this study has been programmed and executed on the same comput-

er system to ensure using the same hardware and software specifications.
- Before starting the test process and measurements, all user applications (excluding test application) have been closed.
- During the test process and measurements, the used computer system has been disconnected from the Internet.
- No processing has been performed by the test application on list items. Thus, only the time required to iterate over list items has been considered.
- Every test scenario has been repeated 50 times and measurements have been averaged to ensure accuracy via balancing variations in the run time [12].
- All test results have been recorded after creating lists and their items.

## 2. Test Scenarios

The performance of five list iteration methods is compared experimentally via different test scenarios. Each method used five different list sizes (100, 1000, 10000, 100000, and 1000000 items) and for each list size, three different item sizes (1, 100, and 1000 bytes) have been used. It is worth mentioning that the test scenarios mentioned before have been chosen carefully to cover different aspects of each method's overall performance. Besides, the Java code of each method is presented in Table 1.

**Table 1.** Implementations of list iteration methods

| List iteration method | Java code |
|---|---|
| Classic For | for( int i = 0 ; i < listSize ; i++ )<br>list.get( i ); |
| Enhanced For | for( String s : list ) { } |
| Iterator | Iterator<String> iterator = list.iterator ( );<br>while( iterator.hasNext ( ) )<br>iterator.next ( ); |
| List Iterator | Iterator<String> listIterator = list.listIterator ( );<br>while( listIterator.hasNext ( ) )<br>listIterator.next ( ); |
| For Each | list.forEach( iter -> { } ); |

## 3. Test Metrics

The time (in milliseconds) required to iterate sequentially over list items by each list iteration method has been used as a metric to evaluate and compare practically the performance of each method. Thus, any list iteration method iterates over items of a list in a less time is considered as the best from a performance point of view.

## 4. Testbed Setup

This study has been setup with software and hardware which their specifications are presented respectively in Table 2 and 3.

**Table 2.** Specifications of software used in the study

| | Software | Version |
|---|---|---|
| **Test application** | Java Development Kit | 1.8.0_161 |
| | NetBeans IDE | 8.2 |
| **Operating System** | Microsoft Windows | 7 Home Basic (64-bit) |

**Table 3.** Specifications of hardware used in the study

| | Hardware | Detail |
|---|---|---|
| **Computer System** | Model | Laptop: ASUS K34S Series |
| | CPU Type | Intel Core i5-2450M |
| | CPU Speed | 2.5 GHz |
| | CPU Cores | 4 |
| | RAM | 6 GB |
| | Rating (Windows Experience Index) | 4.5 |

## EXPERIMENTAL RESULTS

In this section, the experimental results of this study have been presented in Table 4.

**Table 4.** Experimental results of list iteration methods

| List Size (no. of items) | Item Size (byte) | Classic For (ms) | Enhanced For (ms) | Iterator (ms) | List Iterator (ms) | For Each (ms) |
|---|---|---|---|---|---|---|
| 100 | 1 | 0.017 | 0.042 | 0.006 | **0.005** | *6.051* |
| | 100 | 0.021 | 0.049 | 0.007 | **0.006** | *6.351* |
| | 1000 | 0.025 | 0.055 | 0.008 | **0.007** | *6.489* |
| 1000 | 1 | 0.066 | 0.182 | 0.029 | **0.027** | *6.554* |
| | 100 | 0.074 | 0.199 | 0.031 | **0.029** | *6.730* |
| | 1000 | 0.104 | 0.211 | 0.034 | **0.033** | *6.797* |
| 10000 | 1 | 0.436 | 0.809 | **0.153** | 0.169 | *8.286* |
| | 100 | 0.493 | 0.863 | **0.155** | 0.183 | *8.592* |
| | 1000 | 0.532 | 0.959 | **0.158** | 0.189 | *8.646* |
| 100000 | 1 | 0.868 | 1.633 | **0.366** | 0.393 | *9.109* |
| | 100 | 0.913 | 1.654 | **0.373** | 0.403 | *9.297* |
| | 1000 | 0.930 | 2.110 | **0.396** | 0.422 | *9.709* |
| 1000000 | 1 | 1.000 | 2.672 | **0.702** | 0.715 | *10.061* |
| | 100 | 1.015 | 2.803 | **0.843** | 0.854 | *10.530* |
| | 1000 | 1.022 | 2.890 | **0.924** | 0.956 | *10.817* |

From Table 4 (bold represents the better results and italic represents the worst results), the following observations have been indicated:

- Iterator and List Iterator outperformed the other list iteration methods in all test scenarios. The times they require to iterate over list items were very small compared to others.
- List Iterator outperformed Iterator when the list size was small. On the other hand, Iterator outperformed List Iterator when the list size was large. However, the difference in performance between the two methods was very slight.
- For each was the worst one in performance in all test scenarios compared to other list iteration methods. It requires more time to iterate over list items compared to others.

## CONCLUSIONS

This paper presented an experimental approach to evaluate the performance of five list iteration methods in Java in terms of the time required to iterate over list items. Different list sizes and different sizes of list items have been used to achieve the aforementioned goal. Overall performance evaluation showed that Iterator and List Iterator methods outperformed other list iteration methods in all test scenarios. However, List Iterator outperformed Iterator when the list size was small. On the other hand, Iterator outperformed List Iterator when the list size was large. Table 4 presented the summary of this study. This study is crucial to help developers to select a list iteration method with an acceptable level of performance to develop software applications with critical time needs. For the future, some works could be done as: (a) applying the evaluation approach used in this paper to evaluate the list iteration methods with each other but with lists of larger sizes (b) measuring the impact of changing the size of list items on the overall performance of each method (c) measuring the impact of changing the type (e.g. integer, float, or char) of list items on the overall performance of each method.

## REFERENCES


1. Deitel P. and Deitel H., Java™ How to Program, 10th Edition, Pearson, 2015.
2. Sarhan QI, Gawdan IS. Web Applications and Web Services: A Comparative Study. Sci J Univ Zakho [Internet]. 2018;6(1):35–41. Available from: https://doi.org/10.25271/2018.6.1.375
3. Sarhan QI. Internet of things: a survey of challenges and issues. Int J Internet Things Cyber-Assurance [Internet]. 2018;1(1):40. Available from: http://www.inderscience.com/link.php?id=90162 https://doi.org/10.1504/IJITCA.2018.10011246
4. Gupta A, Ashraf M. Comparative analysis of encapsulated Java collection framework based on storage attributes. In: International Conference on Computing, Communication & Automation [Internet]. IEEE; 2015. p. 914–7. Available from: http://ieeexplore.ieee.org/document/7148506/ https://doi.org/10.1109/CCAA.2015.7148506
5. Oracle Website: https://docs.oracle.com/javase/tutorial, accessed 02/09/2018.
6. Togashi N, Klyuev V. Concurrency in Go and Java: Performance analysis. In: 2014 4th IEEE International Conference on Information Science and Technology [Internet]. IEEE; 2014. p. 213–6. Available from: http://ieeexplore.ieee.org/document/6920368/ https://doi.org/10.1109/ICIST.2014.6920368
7. Das S, Kone V. Ruby under Scanner : Comparison with Java, University of California, technical report, 2010. Available from: https://pdfs.semanticscholar.org/8a06/b3a1694b7ee9ac3b2211b7cbc05efc7528ee.pdf
8. Hundt R. Loop Recognition in C ++ / Java / Go / Scala. Proc Scala Days. 2011;1(1):38–47.
9. Wendykier P, Borucki B, Nowinski KS. Large Java arrays and their applications. In: 2015 International Conference on High Performance Computing & Simulation (HPCS) [Internet]. IEEE; 2015. p. 460–7. Available from: http://ieeexplore.ieee.org/document/7237077/ https://doi.org/10.1109/HPCSim.2015.7237077

10. Costa D, Andrzejak A, Seboek J, Lo D. Empirical Study of Usage and Performance of Java Collections. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17 [Internet]. New York, New York, USA: ACM Press; 2017. p. 389–400. Available from: http://dl.acm.org/citation.cfm?doid=3030207.3030221 https://doi.org/10.1145/3030207.3030221
11. Sarhan QI, Gawdan IS. Java Message Service Based Performance Comparison of Apache Activemq and Apache Apollo Brokers. Sci J Univ Zakho [Internet]. 2017;5(4):307–12. Available from: https://doi.org/10.25271/2017.5.4.376
12. Corral-García J, González-Sánchez J-L, Pérez-Toledano M-Á. Evaluation of Strategies for the Development of Efficient Code for Raspberry Pi Devices. Sensors [Internet]. 2018 Nov 21;18(11):4066. Available from: http://www.mdpi.com/1424-8220/18/11/4066 https://doi.org/10.3390/s18114066