



SDI Review Form 1.6

Journal Name:	British Journal of Mathematics & Computer Science
Manuscript Number:	2014_BJMCS_15476
Title of the Manuscript:	Software Design Concerns Associated with Simulating Multiple Inheritance in Java for Implementation Purposes
Type of the Article	Original Research Article

General guideline for Peer Review process:

This journal's peer review policy states that **NO** manuscript should be rejected only on the basis of '**lack of Novelty**', provided the manuscript is scientifically robust and technically sound.

To know the complete guideline for Peer Review process, reviewers are requested to visit this link:

(<http://www.sciencedomain.org/page.php?id=sdi-general-editorial-policy#Peer-Review-Guideline>)



SDI Review Form 1.6

PART 1: Review Comments

	Reviewer's comment	Author's comment (if agreed with reviewer, correct the manuscript and highlight that part in the manuscript. It is mandatory that authors should write his/her feedback here)
<u>Compulsory</u> REVISION comments	The following is my review of the manuscript, "Software Design Concerns Associated with Simulating Multiple Inheritance in Java for Implementation Purposes". The technical components of the paper are sound and the manuscript observed that, OO software development has become the dominant development approach with Java as the common implementation language. In the author's paper, they addresses important aspects related to implementing multiple inheritance, in Java, that may be neglected by developers, and proposes two additional steps in the development life cycle when implementing a system. The paper makes a valuable contribution to the literature and it is of value to the readers of "BJMSC". However, the authors have NOT assembled a relevant and rigorous literature review. Although some of them are seminal, there are only twelve references. While there are two 2014 references and one 2013 reference, the next newest references is in 2009. It does not demonstrate an adequate understanding of the relevant literature, in terms of making a connection between Multiple Inheritance in Java for Implementation Purposes and this paper, which is rather weak. In other words, the authors need	Thank you for your feedback with regards to our submitted manuscript. We found your comments highly valuable. The list of readings that you provided us is very informative which encouraged us to critically read each. As a result we found the list of readings shown below to be relevant to the work described in our submission and of great value; therefore we modified our text where appropriate to strength our arguments referring to them. E. Nasser, S. Counsell, and M. Shepperd. Class movement and re-location: An empirical study of Java inheritance evolution. The Journal of Systems and Software, 2010: volume 83, pages 303-315. Published by Elsevier Inc. Giovanni Lagorio, Marco Servetto, and Elena Zucca. Featherweight Jigsaw – Replacing inheritance by composition in



SDI Review Form 1.6

	to clearly discuss their research contributions to Software Design Concerns Associated with Simulating Multiple Inheritance related work. For example, there is no mention of the "diamond problem" found in Java 8, because prior to Java 8, Java was not subject to the Diamond problem risk, as Java did not support multiple inheritances. However, the Java 8 compiler provides rules to determine which default method a particular class uses, which prevents the Diamond problem. I would suggest that they investigate and try to incorporate the following references, as well as others, that would assist them in contrasting, investigating and incorporating a similar Java multiple inheritances into their Software Design approach: Class movement and re-location: An empirical study of Java inheritance evolution, Journal of Systems and Software, Volume 83, Issue 2, February 2010, Pages 303-315, E. Nasser, S. Counsell, M. Shepperd Featherweight Jigsaw — Replacing inheritance by composition in Java-like languages, Information and Computation, Volume 214, May 2012, Pages 86-111, Giovanni Lagorio, Marco Servetto, Elena Zucca On flexible dynamic trait replacement for Java-like languages, Science of Computer Programming, Volume 78, Issue 7, 1 July 2013, Pages 907-932, Lorenzo Bettini, Sara Capecchi, Ferruccio Damiani Separation Logic for Multiple Inheritance, Electronic Notes in Theoretical Computer Science, Volume 212, 30 April 2008, Pages 27-40, Chenguang Luo, Shengchao Qin Martin, Robert C. (1997-03-09), Java and C++: A critical	Java-like languages. Information and Computation, 2012: volume 214, pages 86-111. Published by Elsevier Inc. Oracle 2014. Multiple Inheritance of State, Implementation, and Type. Accessed 15th December 2014. Available: http://docs.oracle.com/javase/tutorial/java/landl/multipleinheritance.html.
--	---	---



SDI Review Form 1.6

	comparison http://cr.openjdk.java.net/~briangoetz/lambda/lambda-state-final.html http://docs.oracle.com/javase/tutorial/java/landI/multipleinheritance.html	
<u>Minor</u> REVISION comments	The previously mentioned strengths of the manuscript suggest that the paper can be considered to be of high quality and also be of sufficient value to the readership of "BJMSC". Therefore, based on the foregoing discussion, I would recommend that this paper be accepted, after reviewing and integrating the basic concepts and the citations, which are referred to above and after updating the author's literature review.	Completed as discussed above.
<u>Optional/General</u> comments	The following is my review of the manuscript, "Software Design Concerns Associated with Simulating Multiple Inheritance in Java for Implementation Purposes". The technical components of the paper are sound and the manuscript observed that, OO software development has become the dominant development approach with Java as the common implementation language. In the author's paper, they addresses important aspects related to implementing multiple inheritance, in Java, that may be neglected by developers, and proposes two additional steps in the development life cycle when implementing a system.	