Design and Development of a Procedure for new Object-Oriented Design Metrics

K.P. Srinivasan
Assistant Professor of Computer Science
C. B. M. College
Kovaipudur, Coimbatore-641042, India

Dr T. Devi
Reader and Head
Department of Computer Applications
Bharathiar University Coimbatore -641046, India

ABSTRACT
This paper introduces a procedure that has been developed for evaluating an object-oriented design of a system that involves many classes. This approach involves two new metrics called Total Class Metric (TCM) and Total System Metric (TSM) that assess the design of a class and system as a whole respectively during object-oriented development process. In the increasing use of object-orientation in software development, there is a growing need to measure efficiency and effectiveness of the design process. In response to this need, a number of researchers have developed various metrics for object-oriented systems. A procedure has been introduced for evaluating the effectiveness of the object-oriented design of a system for the improvement of the software process instead of using individual design metrics. The total class metric is defined based on a set of seven metrics which have been formulated using main attributes and significant characteristics of an object-oriented design of the system. This research paper discusses in detail about the new approach, total class metric and total system metric to represent the single quality value for the entire system design to judge the effectiveness of the design. These metrics will be useful in measuring object-oriented design and feedback system of software measurement thus yielding an effective object-oriented design.

General Terms
Software Measurement, Single Overall Metric, Technical Metrics, Procedural Approach

Keywords
Software Metrics, Object-Oriented Metrics, Class Metrics, System Metrics.

1. INTRODUCTION
Object-oriented metrics is a still-evolving field and to improve the Object-Oriented Design (OOD), software measure and metrics are needed. Software metrics may be broadly classified as either product metrics or process metrics. Product metrics are the measures of the software product at any stage of its development. Process metrics on the other hand are measures of the Software development process [1, 5, 6, 7, 15, 22]. For object-oriented software, there are different sets of design metrics are suggested by different groups for different attribute measures [1, 6, 15]. In this research, a procedure has been designed and developed to test the effectiveness of design based on seven metrics and it incorporates most of the significant, important features of object-orientated system. The metric set is defined through the thorough study of object-oriented design metrics available in the literature. This paper newly introduces two overall metrics – a single overall metric for representing the quality of the class called Total Class Metric and another single overall metric to represent the quality of the overall system design called Total System Metric. In this procedural approach will add the confident on the application of software metrics in easy manner and solve the difficulties referred in literature on usefulness, application methodology, easy understanding and result oriented [2, 9, 17, 19]. A set of object-oriented metrics is explained in next section. In section III procedural approach for object-oriented design metrics is discussed in detail. In section IV, case study and illustrative examples are presented and conclusion includes future directions of the research.

2. A SET OF OBJECT-ORIENTED DESIGN METRICS
In the development of software metrics research, during the first decade of the 21th century are really encouraging [3, 13, 14, 17, 19, 20, 22]. The recently proposed software metrics are being applied more widely, with good results in many cases. A set of seven object-oriented metrics have been formulated drawing upon the most significant characteristics of object-orientation. These metrics will get the values in an easy way, when apply it in object-oriented design. This metric set will be more comprehensive, complete and quickly measure the characteristics of object- oriented design. A set of seven metrics are defined and explained below with comparison of three main groups in this research field, namely, Chidamber and Kemerer [5-8, 12, 21] Brito e Abreu [1,11,16,23] and Lorenz and Kidd[15, 24].

Metric 1: Methods-Per-Class Factor (MPCF)

$$MPCF = \frac{MPC}{M_{\max}}$$

Method Per Class (MPC) is the number of methods excluding inherited methods defined in the class and M_{max} is the maximum number of methods that may be allowed in a class. Since influence of the inherited methods is taken into account later in the MIF metric (Metric 4), they are not included in the count for MPCF. This stand is similar to that of Chidamber and Kemerer [5-7, 12, 21], but different from the stand taken by Lorenz and Kidd [15, 16, 22]. MPC value of 20 is recommended by Lorenz through experience [15].

Metric 2: Attributes-Per-Class Factor (APCF)

$$APCF = \frac{APC}{A_{\max}}$$

Attributes Per Class (APC) is the number of attributes excluding inherited attributes defined in the class and A_{max} is the maximum number of attributes that may be allowed in a class. Since influence of the inherited attributes is taken into account later in the AIF metric (Metric 5), they are not included in the count for APCF. According to Lorenz and Kidd [15, 16, 24], class size is determined by the total number of attributes and methods in a class. To measure the class size, attributes are also considered equally like MPC [3]. APC value of 6 is recommended by Lorenz through experience [15].

Metric 3: Depth-of-Inheritance-Level Factor (DILF)

$$DILF = \frac{DIL}{D_{\max}}$$

Depth of Inheritance Level (DIL) of a class is the maximum length from that class to the root of the class hierarchy. D_{max} is the maximum number of inheritance levels allowable in class hierarchy. Lorenz recommended DIL value of 6 through experience [15]. Depth of Inheritance Level is similar to that of Chidamber and Kemerer metrics [6, 21]. DILF is selected due to it has greater complexity associated with it and key feature of object-oriented design [13, 17].

Metric 4: Method Inheritance Factor (MIF)

$$MIF = \frac{NIM}{NDM + NIM}$$

The Method Inheritance Factor (MIF) is defined as the ratio of the Number of Inherited Methods (NIM) to the Number of Defined Methods(NDM) and inherited methods in the class. MIF is similar to that of Brito e Abreu metrics called as Metrics for Object-Oriented Design (MOOD) [1, 11, 23].

Metric 5: Attribute Inheritance Factor (AIF)

$$AIF = \frac{NIA}{NDA + NIA}$$

The Attribute Inheritance Factor (AIF) is defined as the ratio of the Number of Inherited Attributes (NIA) to the Number of Defined Attributes(NDA) and inherited attributes in the class. AIF is similar to that of MOOD metrics [1, 11, 23].

Metric 6: Coupling Factor (CF)

$$CF = \frac{NAC}{NPC}$$

NAC is the Number of Actual Couplings with other classes and NPC is the Number of Possible Couplings of this class with other classes of the system. Clearly, the number of possible couplings of a class with other classes of the system equals (Number of classes – 1).

Coupling Factor (CF) for a class is defined as Number of other classes to which coupled / (Number of classes – 1). Since inheritance is already considered in DILF (Metric 3), MIF (Metric 4) and AIF (Metric 5) metrics, inheritance is excluded in determining the couplings [17, 21, 22].

Metric 7: Lack-of-Cohesion Factor (LCF)

$$LCF = \frac{NDM}{NPM}$$

NDM is the Number of Dissimilar Method pairs in the class and NPM is the Number of Possible Method pairs in the class. If two methods access one or more common attributes, then these two methods are similar. And if two methods have no commonly accessed attribute then these two methods are dissimilar. When there are many similar method pairs in a class, then there is good cohesion in the class. Lack of cohesion defined as if m is the number of methods in the class, then the number of possible method pair is m (m-1)/2. The definition of LCF given above is different from the definition of Lack of Cohesion in Methods metric of Chidamber and Kemerer [6, 7, 19]. The set of seven metrics are validated using Weyuker's properties of measures. Next section explains the procedural approach for object-oriented design metrics.

3. A TOTAL CLASS AND SYSTEM METRICS FOR OBJECT-ORIENTED DESIGNS

Metrics are appreciated only when they are clearly needed and easy to collect and clearly understood. Most metrics defined and used are stand alone metrics for measurement. In order to improve the quality and productivity of software, organizations integrated the measurement and process activity. Current techniques in industrial environment adopt measurement based process improvement [13, 19, 22]. The design experts of a particular domain can design a formal object-oriented design for the software development in order to produce high quality software [8, 19]. To find the effectiveness of the object-oriented design, a procedural approach has been suggested here and execution of each and every step is detailed. This procedure yields a single metric value of the called Total Class Metric and yields single metric value for a system called Total System Metric for the entire system.

Fig. 1 shows the procedural approach for object-oriented design metrics. This approach will overcome the problems in application of metrics and obtains the values from metrics. The execution and methodology of each step are detailed below. In step 1, selection of a metrics or metric set to measure the attributes of object-oriented design is based on the designer or user to select the attributes to measure based their current needs or usage of metrics in their project development process or product. In step 2, metrics or metric set may be formed using any one of the following methodology: (i) develop a new metrics set (ii) use an already available metrics set (iii) develop an modified metric set from currently available metric suite for a procedure approach. Here, a metric set has been formulated drawing upon the most significant characteristics of object-oriented development model of the domain usage. In step 3, calculate the values of the defined metrics set as detailed in section II. In step 4, calculate the values of the defined metrics set and tabulate the values for all classes of the system.

Step 1: Select a metrics or metric set to measure the attributes of object-oriented design.

Step 2: Form a metric set for a procedure to measure the effectiveness of an object-oriented design of a particular domain.

Step 3: Calculate and obtain the attribute values of the metric set.

Step 4: Obtained metrics values are tabulate for all classes of the system for easy usage and manipulation of metric data.

Step 5: Calculate the Total Class Metric value using the defined metrics formula for TCM as shown below and tabulate their values for the all classes in the table used in step 4.

Let μ_i denote the i^{th} metric and 'η' the total number of metrics in the metric suite. Total Class Metric formula can be defined as:

$$TCM = \frac{\sum_{i=1}^{\eta} \mu_i}{\eta} where i = \eta$$

Step 6: Obtain the Total System Metric value using the defined metrics formula for TSM as shown below and find the values for the system. TSM value is useful to have a single quantity representing the quality of design of the entire system.

Let TCM_j is the Total Class Metric of class j and c is the total number of classes in the system and Total System Metric formula can be defined as:

$$TSM = \frac{\sum_{j=1}^{c} TCM_j}{c} where j = c$$

Step 7: Closely examine the TCM values of class and TSM values of the system. If class have a highest value that is TCM > 1 and TSM > 1 then modify the classes and apply the step 3 to step 7 until TCM < 1 and TSM < 1 in order to improve the effectiveness of design.

Step 8: If TCM and TSM values are very high due to more number of classes for that domain or environment, then, apply and check the Accepted Metric Domain Value (AMDV). (Here, AMDV=1). If TSM <= AMDV then design will consider for project. If TSM > AMDV then design will consider for Definite Reject for a particular domain of a project.

Step 9: If the designer requires more accuracy of a metrics value or check with other metrics of a particular attributes = YES ,then, execute to Attribute Metric Manager for corresponding metrics defined by others and it give values of attributes defined by other metrics approach. (As per software metrics rules, any attributes of design can measure in different dimensions, methods, approach in software measurement field and different metrics are allowed for particular attributes.)

Step 10: If TCM and TSM values are good with step 7 to step 9 then that project design is effective design of a particular domain. If TCM and TSM values are not correlated with AMDV values then that design will reject for project.

Fig. 1 New Procedural Approach for Object-Oriented Design Metrics

In step 5, calculate the Total Class Metric value using the defined metrics formula for TCM as shown in step 5 and tabulate the values for the all classes of the system in the table used in step 4. In step 6, obtain the Total System Metric value using the defined metrics formula for TSM as shown in step 6 and find the values for the system. In step 7, closely examine the TCM values of class and TSM values of the system. Normally, a good system design will result in a low value for the total metric value of system. If a system has a high value for TSM then the design needs to be revised and improved. This step is used for evaluating the effectiveness of design based on the threshold values which are defined by the design experts using domain environments and applications. Normally, below the average value of TSM is acceptably good. In Step 8, newly introduces the Accepted Metric Domain Value (AMDV) manager. If TCM and TSM values are very high due to more number of classes for that domain or environment, then, apply and check the Accepted Metric Domain Value (AMDV) manager: if TSM <= AMDV then design will consider for project or if TSM > AMDV then design will consider for Definite Reject for a particular domain of a project. In step 9, check with an Attribute Metrics Manager of the feedback system in software measurement as shown in the Fig. 2. This step gives to the designer of a system required more accuracy of a metrics value or check with other metrics of a particular attributes. Step 10, the feedback system compares the result of the steps from step 7 to step 9 of a procedure and finally produces the TCM and TSM values from obtained metrics values. In order to choose the efficient design from among competing designers that is, the system designed by a few design experts in large projects, the method described above is also used. In order to find the efficient design among many designs, TSM values corresponding to different designs are calculated first and the lowest value among the TSM values of different designs is chosen. The design corresponding to the lowest TSM value is an efficient design among many designs.

4. ILLUSTRATIVE EXAMPLES

This proposed approach for object-oriented design metrics is also used in feedback system of software measurement field. This section illustrates the use of a set of object-oriented design metrics and the total class and system metrics in procedural approach. An illustrative example called the Trader system is given here [5]. This class system is shown in Fig. 3.

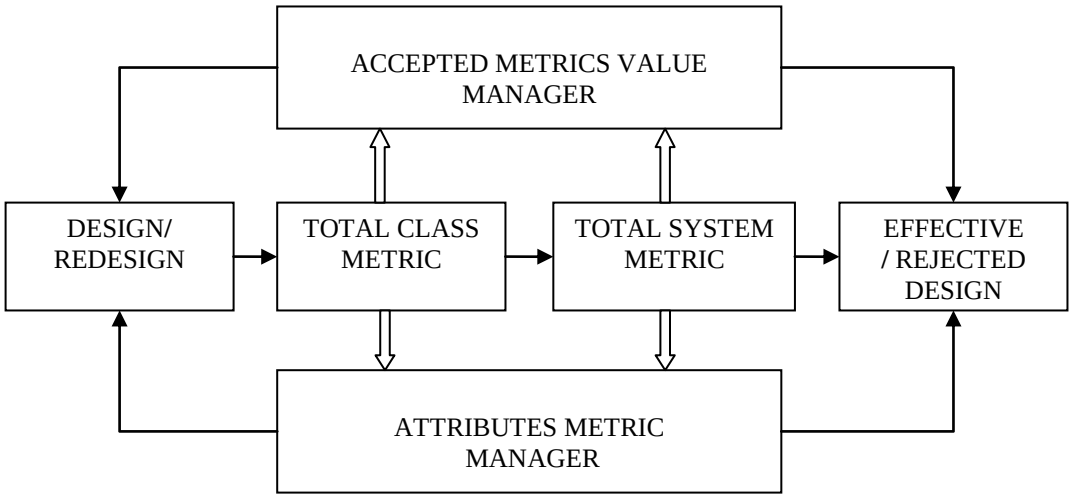

Fig. 2 Feedback Systems in Software Measurement

TABLE 1 TOTAL CLASS METRIC OF THE TRADER SYSTEM

Class Name	MPCF	APCF	DILF	MIF	AIF	CF	LCF	∑μι	TCM
TRADE	0.15	0.5	0	0	0	0.14	0.67	1.46	0.21
BOND	0.05	0.17	0.17	0.75	0.75	0	0	1.89	0.27
FX	0.05	0.17	0.17	0.75	0.75	0	0	1.89	0.27
EQUITY	0.1	0.83	0.17	0.6	0.38	0.28	0	2.36	0.34
MUNICIPAL	0.05	0.33	0.33	0.8	0.66	0.14	0	2.31	0.33
CORPORATE	0.05	0.33	0.33	0.8	0.66	0.14	0	2.31	0.33
INTERNATIONAL	0.1	0.33	0.33	0.71	0.8	0.14	1	3.41	0.49
DOMESTIC	0	0.17	0.33	1.0	0.88	0	0	2.38	0.34
THE SUM OF TCM METRICS									2.58

The attributes and methods of various classes are given as pseudo code in Appendix 1.For illustration of the application of object-oriented design metrics, consider the class, International Trade. For this class the number of methods defined in the class is 2 and inherited methods are 5. Taking M_{max} = 20, MPCF = 2/20=0.1. And, MIF = 5/2+5 = 5/7 = 0.71. Attributes defined in the class are 2, and inherited attributes are 8. Taking A_{max}=6, APCF = 2/ 6 =0.33. And, AIF = 8/2+8=8/10=0.8.

In Fig. 3 it is seen that International Trade class is two levels below the root class. Taking DIL_{max}=6, DILF = 2/6=0.33. This class is calling the *calculate-exchange-rates()* function of the class FX Trade. There is no other interaction with any other class. The number of classes in this system is 8. Hence CF = 1/8-1=1/7=0.14. There are two methods and two attributes defined in the class. No attribute is accessed by the both the methods. Hence dissimilar pair is 1. Hence LCF = 1/(2(2-1)/2) = 1/(2x1/2) = 1/1=1. For this class Total Class Metric is [0.1+0.33+0.33+0.71+0.8+0.14+1]/7 = 3.41/7=0.49. Metrics for other classes are calculated in a similar manner and given in Table 1.

The sum of the total class metric of all the 8 classes is 2.58. Hence the total system metric for the Trader system is 2.58/8 = 0.32. Perusal of the total class metrics of individual classes shows that they are low enough and hence their designs may be accepted as good. The value of total system metric is low enough and hence the design of the system is judged as acceptably good. The total class metric and total system metric may be used in guiding the design of individual classes and used in feedback system in software measurement. Good design of a class should result in a low value for the total class metric. If any

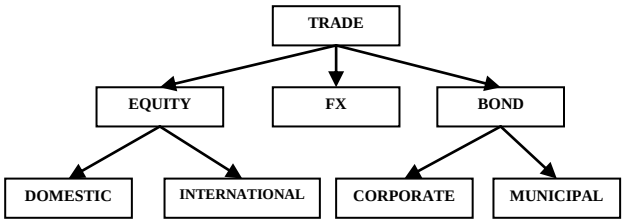

Fig. 3 Class Hierarchy for the Trader System

class design gives rise to a large value then that class needs to be more closely examined and redesigned better . Similarly, a good system design will result in a low value for the total system metric. If a system has a high value for TSM then the design needs to be revised and improved. Threshold values for TCM and TSM are considered good may be selected by the design team based on previous experience in objected-oriented design of the systems.

5. CONCLUSIONS

This paper introduces a procedural approach for single overall metric called total class metric and total system metric for a system consisting of many classes for evaluating the object-oriented design of a system. In this research paper, also introduces the usage of metrics as a feedback system in software measurement field to measure the software design. A set of seven metrics also proposed for the important features of object-orientation. This procedural approach gives the values of the total metric value of system for judging the quality of object-oriented design of entire system. The same approach is also useful for selection of efficient design among many designs. Application of the seven metrics and the total class and system metrics has been illustrated through an illustrative example. These metrics will be useful in guiding the design of object-oriented systems. This feedback system, total class metric and total system metric of software measurement work can be further extended to measure the software Process Efficiency and Product Effectiveness (PEPE) for the development of software.

6. REFERENCES

[1] Abreu, F.B., Melo, .W. 1996. Evaluating the impact of OO Design on Software Quality. Proceeding of Third International Software metrics Symposium, Berlin, German.

[2] Amjan Shaik., Reddy, C.R.K., Manda,B., Prakashini, C., Deepthi, K. 2010. Metrics for Object Oriented Design Software Systems: A Survey. Journal of Emerging Trends in Engineering and Applied Sciences. 1(2): 189-197.

[3] Bandi, R.K, Vaishnavi, V.K., Daniel, E Turk. 2003. Predicting Maintenance Performance Using Object - Oriented Design Complexity Metrics", IEEE Transactions on software Engineering. vol. 29. no. 1. 77-89.

[4] Basili, V.R., Briand, L.C., Melo, W.L. 1996. A Validation of Object–Oriented Design Metrics as Quality Indicators. IEEE Trans. on soft. Eng. vol. 22, no.10. 751-761.

[5] Chidamber, S.R., Darcy, D.P., Kemerer, C.F. 1998. Managerial use Of Metrics for Object- Oriented Software: An Exploratory Analysis. IEEE Transactions on Software Engineering. vol.24, no. 8. 629-639.

[6] Chidamber, S.R., Kemerer, C.F. 1994. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering. vol. 20.no. 6. 476-493.

[7] Churcher, N.I., Sheppard, J. 1995. Comments on 'A Metrics Suite for Object– Oriented designs. IEEE Transactions on software Engineering. vol. 21. no. 3. .263-265.

[8] Devi, T., Srinivasan, K.P. 2010. Statistical Techniques in Software Quality Measurement and Metrics. Proceeding of UGC Funded National Conference on Recent Advances in Statistics and Computer Applications, Bharathiar University, Coimbatore.

[9] Gurdev Singh, Dilbag Singh, Vikram Singh, P.M. 2011. A Study of Software Metrics. International Journal of Computational Engineering and Management. vol. 11. 2230-7893.

[10] Emam, K E., Benlarbi, S., Goel, N., Rai, S N . 2001. The Confounding Effect of Class Size on the Validity of Object Oriented Metrics. IEEE Transactions on Software Engineering. vol. 27.no.7..630-650.

[11] Harrison, R., Counsell, S.J., Nithi, R.V. 1998. An Evaluation of the MOOD Set of Object- Oriented Software Metrics. IEEE Transactions of Software Engineering. vol. 24. no.6. 491-496.

[12] Hitz, M., Montazeri, B., C.K. Metrics Suite: A Measurement Theory Perspective. IEEE Transactions on software Engineering. vol. 22, no. 4.267- 271.

[13] Kumar Rajnish, A.K., Choudhary., Agawal, A.M. 2010. Inheritance Metrics For Object-Oriented Design. International Journal of Computer Science and Information Technology. vol.2. no.6. 13-25.

[14] Linda Badri, M, Badri., F, Toure. 2011. An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes. International Journal of. Software Engineering and Its Applications Vol. 5 No. 2. 69-85.

[15] Mark Lorenz. 1993. Object-Oriented Software Development: A Practical guide. 1993. Prentice hall, Englewood Cliffs, New Jersey.

[16] Pressman R.S. 2005. Software Engineering A Practitioner Approach. 6thEd, McGraw-Hill.

[17] Rakesh Kumar., Gurvinder Kaur. 2011. Comparing Complexity in Accordance with Object Oriented Metrics. International Journal of Computer Applications. . Volume 15– No.8.42-45 .

[18] Sarkar, S., KaK, A., C,Rama, G.M. 2008. Metrics for measuring the quality of modularization of Large -Scale Object-Oriented Software. IEEE Transactions on Software Engineering. vol. 34. no. 5.700-720.

[19] Shanthi, P.M., Duraiswamy, K. 2011. An Emprical Validation of Software Quality Metrics Suites on Open Source Software for Fault-Proneness Prediction in Object Oriented Systems. European Journal of Science and Research.vol.51.no.2. 168-181.

[20] Srinivasan, K.P., Devi, T. 2009. Procedure for Selection of an Efficient Object-Oriented Design. Proceeding of UGC Funded National Conference on Recent trends in Software Engineering, Sullamussalam Science College, Mallapuram, Kerala.25-27. [**Best Paper of the Conference**].

[21] Srinivasan, K.P., Devi, T., Thiagarasu, V. 2009. Analysis of Chidamber-Kemerer Metrics for Object-Oriented Design. Proceeding of National Conference on Emerging trends in Computer Science, Avinasilingam University for Women, Coimbatore.

[22] Srinivasan K.P., Devi, T. 2009. Design and Development of a Procedure to Test the Effectiveness of the Object-Oriented Design. International Journal of Engineering Research and Industrial Applications. ISSN 0974-1518, vol. 2. no.VI. 15-25. [http:// www. Ascent - journals .com/ IJERIA/Vol2.No6/ Paper2.pdf]

[23] Srinivasan, K.P., Devi, T. 2009. A Case Study Approach for Application of MOOD Metrics in Object-Oriented Design. Proceedings of the International Conference on Global Computing and Communications. Hindustan Institute of Technology and Sciences .Padur. Chennai. 77-82. [ISBN: 978-81-8424-543-1]

[24] Stephen H Kan. 2006. Metrics and Models in Software Quality Engineering. Second Edition. Pearson Education.

APPENDIX 1

PSEUDOCODE FOR CLASSES IN TRADER SYSTEM

This appendix gives the details of the object-oriented design of a system called the Trader System, giving the pseudocode of all the classes of the system.

```
// Class trade
Attributes // a.k.a. instance variables
trade id, counter party, trade value // details of trade
operations // a.k.a. methods
evaluate-counterparty ( ) // determines validity of the counter
party from data base file.
get-trade-id ( ) // obtain trade id details from the user
position-update ( )
call position – manager :: report trade() // send message to
position –  manager class furnishing trade value and trade id.
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - - - - -
/ class bond trade
Attributes // a.k.a. instance variables
bond – details // bond rating details
operations // a.k.a. methods
get-bond – infor ( ) // access bond date base for current market
price information.
- - - - -  - - - - - - - - - -  - -  - - - - - - - - - - - - - - - - - - - - - - - -
// class fx trade
Attributes // a.k.a. instance variables
forex-details // foreign market information
operation // a.k.a. methods
calculate-exchange-rates ( ) // access currency market monitor
for later rates
- - - - - - - - - - - - - - -  -- - - - - - - - - - - - - - - - - - - - - - - - - -
// class equity trade
Attributes // a.k.a. instance variables
company, stock market, PE ratio, earnings, 52-week-hi&lo
operation // a.k.a. methods
estimate-beta ( ) // determine risk compared to rest of the
market.
get-stock-quotes ( ) // consult external stock Quotation data base.
        call Quotron :: Quotes ( ) for trade id // get the latest
Quote for the trade.
- - - - - - - - - - - -- -  - - - - - - - - - - - - - - - - - - - - - - - - - - -
// class municipal bond trade
Attributes // a.k.a. instance variables
state-or-federal, over-the-counter
operations // a.k.a. method

calculate-coupon-rate  ( ) // determine the inter-rate for the bond
        call-Tbill-server :: rates ( ) // get the current rate from
the another source
- - -  - - - - - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - - - - -
// class corporate bond trade
Attributes // a.k.a. instance variables
adr, SP-rating
operations // a.k.a. methods
        call-rating (SP-rating) // get the standard // poor rating
        if adr = = TRUE then call fx trade ::
calculate_exchange_rates () // get the exchange rate information
if this is a foreign bond issue.
- - -  - - -- - - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - - - - -
// class international equity
Attributes // a.k.a. instance variables
Exchange-rate, quotation
operation // a.k.a. methods
perform_analysis_roa (fx trade::calculate_exchange_rates ( ) ) //
analyze the stock using the foreign exchange rate.
get_quotron (quotation) // determine the current price of stock
 - - - - - - - - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// class domestic equity
Attributes // a.k.a. instance variables
attribute 1
operation // a.k.a. methods
none defined yet.
```