Asian Journal of Information Technology 9 (2): 98-100, 2010
ISSN: 1682-3915

Developing a Maintainability Metric for Inner Classes

Sim Hui Tee
Faculty of Creative Multimedia, Multimedia University,
63100 Cyberjaya, Selangor, Malaysia

Abstract: Inner classes are widely used in some programming languages, such as Java. Inner classes are known as helper classes which are defined within outer class. They play the role to assist their outer class in performing a specific task. However, extensive use of inner classes may cause difficulty in software maintenance. This research proposes a maintainability metric for inner classes. The maintainability metric provides a guideline for software developers to define maintainable inner classes in an application.

Key words: Software maintainability, inner classes, object-oriented programming, software metric, outer classes, class complexity

INTRODUCTION

In some of the object-oriented programming languages such as Java, a class could contain nested classes (Sierra and Bates, 2003). These nested classes are known as inner classes (Horstmann and Cornell, 2004). The class which contains inner classes is called outer class (Eckel, 2006). Inner classes are defined as helper classes for their outer class (Sierra and Bates, 2003). The purpose of defining inner classes is to assist outer class in performing a specific task. Thus, inner classes are subsets of outer class which is functionally related to the latter.

Inner classes share all the features of a regular class. They could contain constructors, attributes, methods and further inner classes. Furthermore, an inner class can inherit other class and implement interfaces. Inner classes can be abstract or final as a regular class does (Sierra and Bates, 2003). Program 1 demonstrates a typical inner class named InnerA that is defined within a root outer class named OuterA. Inner class InnerA contains a constructor, attribute, method and another inner class named InnerA1.

Program 1: Examples of inner classes are given:

```
public class OuterA{
    public class InnerA{
    int i;
    public InnerA(){ }
    void run(){ }
    public class InnerA1{
    }
}
}
```

Well-defined inner classes are important to promote class cohesiveness. Unfortunately, extensive use of inner classes increases the difficulty of software maintenance. Software maintenance is a long term effort that involves problem fixing and operational improvement along the course of software evolution (Bhatt *et al*., 2006; Aggarwal *et al*., 2002).

Heitlager *et al*. (2007) claim that the increasing class complexity and class size contribute to increasing maintenance efforts.

Extensive use of inner classes is inevitably leading to greater class complexity and class size, which results in greater difficulty of maintenance.

The feasible solution of this problem is to reduce the number of inner classes that defined within a root outer class. To date, there is no available metric which provides a guideline for software developers in defining maintainable inner classes. This research aims to develop a maintainability metric that helps software developers to understand the extent of maintenance difficulty of inner classes.

This proposed metric provides a guideline for software developers to define maintainable inner classes.

MATERIALS AND METHODS

Maintainability metric for inner classes: The difficulty of maintenance increases as the number of inner classes increases in a root outer class. This research proposes a maintainability Metric (M) as:

$$M = \Sigma c$$

Where:

c = The measure value of each class (outer and inner classes)

$$c = 1/1+n$$

Where:

n = The number of immediate inner class of an outer class

For a regular class that contains no inner class, the M value is 1, as n = 0 and c = 1. When M is 1, it implies that the maintainability of inner class is not an issue because inner class is absent. The higher the M value, the difficulty level of inner class maintainability increases correspondingly.

Program 2 and 3 demonstrate two independent regular classes that have the same difficulty level of maintenance in term of inner class, which is M = 1. Inner classes are absent in Program 2 and 3. Hence, n = 0. House class and Store class have respective, c = 1 based on the above mentioned formula.

Program 2: When summing up c value in Program 2, Class house with M = 1 and 3, respectively M = 1 is yielded. Class house with M = 1 is as:

```
public class House{
   private int size;
      int getSize( ){
         return size;
   }
}
```

Program 3: Class Store with M = 1 is as:

```
public class Store{
}
```

The minimum maintainability Metric value (M) for a class is 1 which indicates that there is no inner class defined. Program 2 and 3 yield the minimum maintainability metric value. When inner class is defined, M value will be >1.

Program 4 demonstrates an outer class MyOuter which contains an inner class MyInner. The number of immediate inner class (n) of MyOuter is 1. The number of immediate inner class (n) of MyInner is 0. Hence, c value for MyOuter is 0.5; whereas c value for *MyInner* is 1. The sum of c values indicates the maintainability Metric value (M), which is 1.5.

Program 4: Root outer class MyOuter with M = 1.5 is as:

```
public class MyOuter{
      public class MyInner{
      }
}
```

For a root outer class which contains multiple inner classes at multiple depth, the c value of all classes needs to be summed up to yield M value for that program. Program 5 shows an example of multiple-depth inner classes. In Program 5 root outer class A contains multiple-depth inner classes is as:

```
class A{
   class B{
               class C{
               }
               class D{
               }
   }
}
```

In Program 5, class A contains an immediate inner class named B. Class B contains two immediate inner classes named C and D, respectively. Inner classes C and D do not contain any inner class. The c values for all classes are shown in Table 1.

Table 1: c value for outer and inner classes

Classes	Number of immediate inner class, n	c
A	1	0.50
B	2	0.33
C	0	1.00
D	0	1.00

Table 2: Summary of maintainability Metric value (M) for four programs

Program	Total number of inner classes	Maintainability Metric value (M)
2	0	1.00
3	0	1.00
4	1	1.50
5	3	2.83

According to Table 1, when c values of all classes are summed up, the maintainability metric value (M = 2.83) is obtained. The maintainability metric values (M) of Program 2-5 summarized in Table 2.

RESULTS AND DISCUSSION

It is observed that as the total number of inner class increases in a program, the maintainability metric value for inner classes is increased. It is because the maintainability metric value is derived from the sum of c values of all classes in that program. Greater maintainability metric value indicates that the difficulty level of inner class maintenance is increased. Based on Table 2, Program 5 has the greatest total number of inner classes and maintainability metric value. It implies that inner classes in Program 5 are harder to maintain as compared to Program 2-4.

Using maintainability metric in defining inner classes: Maintainability metric is a guideline for software developers to decide how many inner classes to be defined within a root outer class. There is no absolute maintainability metric value to decide the total number of inner classes should be defined in a root outer class. The

decision lies in the functionality of a root outer class and its usage demand in the whole application. A root outer class which is expected to perform a detailed functionality can tolerate with a greater maintainability metric value for its inner class. However, a more general class, such as abstract class is not expected to have greater maintainability metric value. It is because greater maintainability metric value renders a general class harder to be maintained as the software evolves.

CONCLUSION

The proposed maintainability metric for inner classes is an indicator of the complexity of inner classes in a program.

A higher maintainability metric value implies that greater maintenance effort is needed to maintain the inner classes. It is desirable to keep the total number of inner classes within a reasonable range. Maintainability metric allows the software developers to reduce the extensive number of inner classes in a root outer class.

REFERENCES

Aggarwal, K., Y. Singh and J. Chhabra, 2002. An integrated measure of software maintainability. Proceedings of Annual Reliability and Maintainability Symposium, Jan. 28-31, Seattle WA., pp: 235-241.

Bhatt, P., K. Williams, G. Shroff and A. Misra, 2006. Influencing factors in outsourced software maintenance. ACM SIGSOFT Software Eng. Notes, 31: 1-6.

Eckel, B., 2006. Thinking in Java. Prentice Hall, USA.

Heitlager, I., T. Kuipers and J. Visser, 2007. A practical model for measuring maintainability. Proceedings of 6th International Conference on the Quality of Information and Communications Technology, Sept. 12-14, Lisbon, Portugal, pp: 30-39.

Horstmann, C. and G. Cornell, 2004. Core Java 2. Prentice Hall, USA.

Sierra, K. and B. Bates, 2003. Sun Certified Programmer and Developer for Java 2 Study Guide. McGraw-Hill, USA.