Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Applied Sciences
5(4), 2017, 166-171

ISSN 2321 – 7456

Original Research Article

CODE REUSABILITY WITH ASPECT USING OBJECT ORIENTED PROGRAMMING LANGUAGE

Shrikant Patel

BPIBS, Delhi, India

Abstract: A standout amongst the significant profits of Object-Oriented programming is that it helps legacy similarly as an instrument to code reuse incremental adjustment, Well-established. Object-oriented innovations in white-box frameworks aggravate exhaustively use. Aspect-Oriented modifying (AOP) may be another technology, which backs an additional development instrument that's intentional may be empower the superior detachment of worries. Across the world useful perspective dialects in Aspect would expand upon object-oriented modifying languages, and in that route it will be conceivable with consolidate both development instruments. This paper reveals the implementation of aspect in object oriented programming language to overcome the problem of cross cutting concerns and aspects are used to less memory occupancy and to increase the processing speed.

Keyword: *AOP, OOP, aspect, cross- cutting, Aspect, Rapid development.*

Introduction: Writing software is a complex business - not only do you have to get the enterprise logic of the application correct, typically you also have to deal with multiple other concerns at the same time, such as "what should happen if something goes wrong", "how should I make sure we know what is happening during execution", "how to enforce security throughout my application" and in some languages "how do I handle memory" or "when should I free up memory", etc.

One of the principal advantages of object-oriented programming[1] techniques over procedural programming techniques is that they enable programmers to create modules that do not require to be changed when a new kind of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify. Object-oriented programming that uses classes is from time to time called class-based programming, while prototype-based programming does not typically use classes. As a result, a significantly dissimilar yet equivalent

For Correspondence:
patelshrikant@rediffmail.com.
Received on: October 2017
Accepted after revision: December 2017
Downloaded from: www.johronline.com

terminology is used to define the concept of object and instance. This paper will solve some problems of software test by introducing a new programming thought named AOP (Aspect Oriented Programming). An aspect is a ordinary feature that's typically scattered across methods, classes, object hierarchies, or even entire object models. In AOP, a feature like metrics is called a crosscutting concern, as it's a behavior that "cuts" across numerous points in your object models, yet is specifically different. As a development style, AOP recommends that you abstract and sum up crosscutting concerns.

Problems caused by cross cutting concerns:

Code Tangling: Each module implements one fundamental concern and parts of other cross-cutting concerns.

Code Scattering: Each cross-cutting concern is implemented by more than a few fragments placed in dissimilar parts of the system.

Consequences:

Low quality: code tangling facilitates errors.

Low traceability: where is the code that implements a given concern? Which concern is implemented by this code?

Low reusability: each module includes fragments that are not related to its job.

Low evolvability: of modules that are fragmented and of modules that include fragments of other modules.

Aspect Oriented Programming: In registering, aspect oriented Programming (AOP) is a programming worldview that plans to expand seclusion by permitting the detachment of cross-cutting concerns. It does as such by adding extra conduct to existing code (guidance) without changing the code itself; rather independently determining which code is altered through a "pointcut" determination, for example, "log all capacity calls when the capacity's name starts with 'set'. This permits practices that are not vital to the business rationale, (for example, logging) to be added to a program without jumbling the code center to the usefulness. AOP shapes a reason for perspective situated programming improvement. Aspect Oriented Programming helps overcome system level coding i.e. Logging, Transaction or Security management problem by centralizing these cross-cutting concerns.

Aspect Oriented Programming tends to every angle independently in a measured manner with insignificant coupling and duplication of code. This particular approach additionally advances code reuse by utilizing a business rationale worry with a different logger aspect.

.

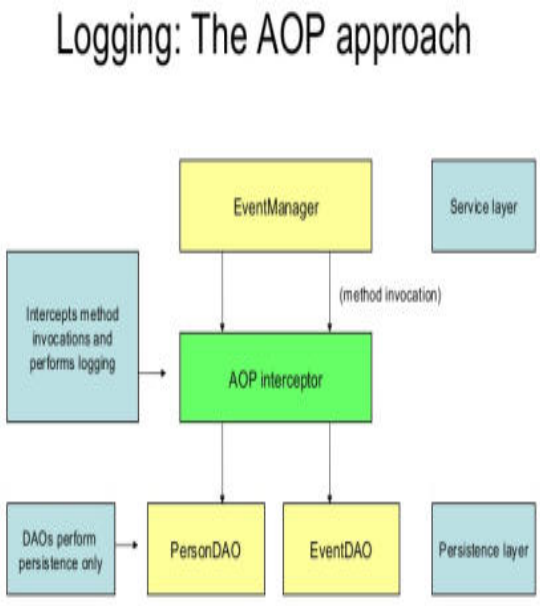

Fig. 1 (a)

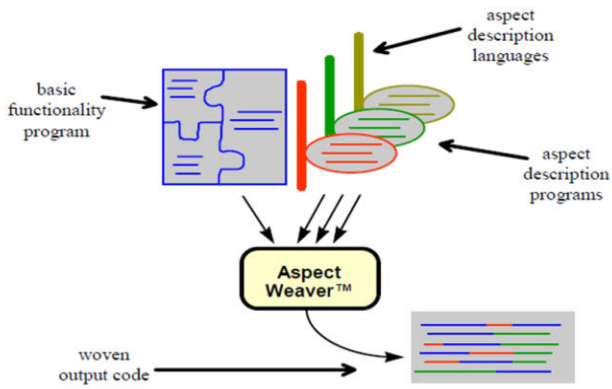

Fig 1 (b)

Fast improvement of transformative models utilizing OOP by concentrating just on the business rationale by excluding cross-cutting

concerns, for example, security, exchange, logging and so on. Once the model is acknowledged, extra concerns like security, logging, evaluating and so on can be meshed into the model code to move it into a creation standard application.

Developers can concentrate on one aspect at a time rather than having to think simultaneously about business logic, security, logging, performance, multithread safety etc. Different aspects can be developed by different developers based on their key strengths.

Aspect J (Aspect Implementation for Java): AspectJ[2] is presently the most complete accomplishment of an AOP language for Java. In practical terms, it is an expansion to Java that treats AOP concepts as first-class elements of the language. It allows aspects to be implemented as part of a Java based application using known Eclipse based tools.

As AspectJ is an extension to Java, Java is the implementation language for the constructs that comprise an Aspect. That is, you use Java to implement whatever behavior the Aspect should provide. A set of rules is then used to determine how to weave the aspect into the main body of your Java application. These rules are implemented by pointcuts, join points, and advice. A pointcut specifies what join points there are. In turn, a join point defines where in a Java programs' execution the aspect should be applied and advice is the implementation of what to do at that point.

Furthermore, AspectJ[2] is viewed as Likewise an execution with versatility properties. This characteristic permits programming particular architects to utilize it on extensive scale frameworks with great come about. It aides on accomplish a respectable diminishment done scattered code with a low execution overhead. These effects help that development from claiming AspectJ likewise an usage for AOP. AspectJ[3] meets expectations with those taking after procedure: Firstly, that java code that reflects the benefits of the business rationale for requisition is produced. Secondly, the viewpoints need aid formed previously, differentiate units (source code files) for components for example, join-points, point-cuts Also exhortation.

Implementation with using AspectJ:

We implement aspects using Cybercafe Example where the aspects can configure[4] at only one time and can use automatically when required so that same method only configured and no need to call again and again as in object oriented programming concept, therefore the memory occupied by the program is lesser in aspect oriented programming than object oriented concepts.

CyberCafe file 1: CyberCafe1.java

```
package cybercafe;
import java.util.Scanner;
import java.util.Calendar;
import java.util.GregorianCalendar;
public class CyberCafe {
   public static Scanner Sc= new
Scanner(System.in);
    static int
Starthour,Startmin,Endhour,Endmin;
   static GregorianCalendar cal= new
GregorianCalendar();
   public static void main(String[] args) throws
Exception
 {
  computer client[]=new computer[10];
       int i;
       for(i=0;i<10;i++) {
          client[i]=new computer(i);
       }
       do {
      System.out.println("\t\t\t1.list of free
computers\n\t\t\t2.assign computers by
id\n\t\t\t3.logout System by id\n\t\t\t4.exit");
       int num;
       num=Sc.nextInt();
       switch(num)  {
          case 1:{
             for(int j=0;j<10;j++)
             {
              if(client[j].check==true)   {
```

```
                System.out.println("pc no.:->"+j);
             }
            }
            break;
          }
          case 2:
           {
            for(int j=0;j<10;j++)    {
             if(client[j].check==true)    {
               System.out.println("pc no.:->"+j);
             }
            }
            System.out.println("Enter no of pc to
assign....");
            int assign=Sc.nextInt();

Starthour=cal.get(Calendar.HOUR_OF_DAY);
            Startmin=cal.get(Calendar.MINUTE);

client[assign].login(Starthour,Startmin);
            break;
          }
          case 3:  {
            for(int j=0;j<10;j++) {
             if(!client[j].check==true)  {
               System.out.println("pc no.:->"+j);
             }
            }
            System.out.println("Enter no of pc to
free....");
            int free=Sc.nextInt();

Endhour=cal.get(Calendar.HOUR_OF_DAY);
            Endmin=cal.get(Calendar.MINUTE);
            client[free].logout(Endhour,Endmin);
            break;
          }
          case 4: {
            System.exit(0);
            break;
          }
          default :   {
            System.out.println("INVALID
INPUT....");
          }
        }
        }while(true);
    }
}
```

CyberCafe file 2:CyberCafe2.java

```
package cybercafe;
import static cybercafe.CyberCafe.Sc;
public class computer {
    boolean check=true;
    int id;
    int pass;
    public computer(int id)    {
        this.id=id;
        this.pass=id;
    }
    void login(int sthour,int stmin)    {
        System.out.print("Enter your id ...");
        if(id==Sc.nextInt())  {
            System.out.print("Enter your pass...");
            if(pass== Sc.nextInt())   {
                check=false;
            }
        }
    }
    void logout(int edhour,int edmin)   {
        if(!check)  {
          check=true;
        }
        else  {
            System.out.print("First you login then
Logout....");
        }
    }
}
```

Aspects file 3:CyberCafe.aj

```
package cybercafe;
public aspect charges
{
int
starthour,startmin,endhour,endmin,snetmin,enet
min;
 pointcut captureCallParameter1(int sthour , int
stmin):
call(void  cybercafe.computer.login(int,int))&&
args(sthour,stmin);
```

```
  before(int sthour, int stmin) :
captureCallParameter1(sthour,stmin)  {
      this.starthour=sthour;
      this.startmin=stmin;
      this.snetmin=(starthour*60);
      this.snetmin=snetmin+startmin;
  }
pointcut captureCallParameter2(int edhour , int
edmin):
call(void cybercafe.computer.logout(int,int))&&
args(edhour,edmin);
  before(int edhour, int edmin) :
captureCallParameter2(edhour,edmin)  {
      this.endhour=edhour;
      this.endmin=edmin;
      this.enetmin=(endhour*60);
      this.enetmin=enetmin+endmin;
       System.out.println("hour:->"+(enetmin-
snetmin)/60);
       System.out.println("min:->"+(((enetmin-
snetmin)%60)));
  }
}
```

By using this example we can calculate the time spent by the customer in cyber café and the amount to be paid by customer to the cyber café owner automatically without calling any method and properties.

Significant Benefits of AOP Over Oop: By using above example, the code of java with using aspect gives the reusability of the code automatically when using same strategy repeatedly[3]. Hence aspect implementation in object oriented programming language reuse the code and these are the benefits of using aspects in object oriented programming language-

- AOP aides overcome framework level coding i.e. Logging, Transaction or Security administration issue by unifying these cross-cutting concerns.
- AOP addresses every angle independently in a measured manner with negligible coupling and duplication of code. This particular approach additionally pushes code reuse by utilizing a business rationale concern with a different lumberjack viewpoint.
- Make less demanding to include more current usefulness' by including new viewpoints and weaving tenets and along these lines recovering the last code. This capacity to include fresher usefulness as particular angles empower application originators to defer or concede some configuration choices without the predicament of over planning the application.
- A code snippet would look WITHOUT AOP:

Fig. 2

- And the same code snippet with AOP:

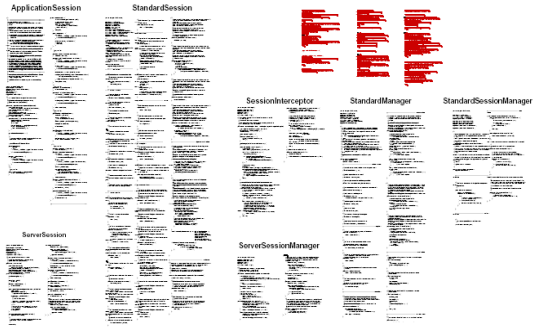

Fig. 3

- Fast improvement of evolutionary models utilizing OOP by centering just on the business rationale by precluding cross-cutting concerns, for example, security, transaction, logging and so forth. Once the model is acknowledged, extra concerns like security, logging, inspecting and so on might be weaved into the model code to move it into a generation standard application.
- Engineers[5] can focus on one perspective at once instead of needing to consider business rationale, security, logging, execution,

multithread wellbeing and so on. Distinctive angles could be produced by diverse engineers focused around their key qualities.

- The cooperation in the middle of perspectives and non-angles in the path depicted above could be acknowledged in AspectJ by assigning all the Exceptions tossed by the advices to the presentation layer and choosing there what to do.

Conclusion: In a decisive manner the key distinction in the middle of OOP and AOP is that the center of OOP is to separate the programming undertaking into items, which embody information and techniques, while the center of AOP is to separate the project into crosscutting concerns. Actually, AOP is not a contender for OOP, in light of the fact that it rose out of OOP standard. AOP develops OOP by tending to few of its issues. AOP acquaints perfect routes with actualize crosscutting concerns[6] (which may have been scattered over a few places in the comparing OOP execution) in a solitary spot.

In AOP, AspectJ is flawless aspect-oriented[6] conservatory to the Java[tm] programming language Java platform[5] well-suited easy to learn and use. It enables clean modularization of crosscutting concerns, such as error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, and multi-object protocols

Consequently, AOP makes the system cleaner and all the more approximately coupled. With expanded IDE help for AOP applications and a developing learning about AOP frameworks the 7). ideal model may well pull in more experts and designers in not so distant future.

References:

1). Hon, T., & Kiczales, G. (2006, October). "Fluid aop join point models". In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 712-713). ACM.

2). Golbeck, R. M., Selby, P., & Kiczales, G. (2010, June). "Late Binding of AspectJ Advice". In *TOOLS (48)* (pp. 173-191).

3). Gulia, P., Dev, A., & Patel, S. (2015, March). "Comparative analysis of object oriented programming and aspect oriented programming approach". In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on* (pp. 1836-1842). IEEE.

4). Hanenberg, S., Bachmendo, B., & Unland, R. (2001). "An object model for general-purpose aspect languages". *Generative and Component-Based Software Engineering*, 80-91.

5). Baca, P., & Vranic, V. (2011, September). "Replacing object-oriented design patterns with intrinsic aspect-oriented design patterns". In *Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the*(pp. 19-26). IEEE.

6). Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., & Irwin, J. (1997). "Aspect-oriented programming". *ECOOP'97—Object-oriented programming*, 220-242.