Abstract—Object Oriented Programming has become one of the most established paradigms. It offers us features like encapsulation, polymorphism, inheritance etc. By using these features we are able to develop good software’s that are easy to understand. But when the size of the software goes on increasing it poses problems with respect to maintaining the source code. Also we come across the problems like code scattering and code tangling. So object oriented programming has some limitations and this is where Aspect Oriented Programming comes into play. AspectJ is one language that makes use of Aspect Oriented Programming. This paper addresses the features and use of AspectJ for Aspect Oriented Programming.

Keywords—Aspect Oriented Programming, AspectJ, Refactoring, Code Scattering, Code Tangling

I. INTRODUCTION

In the initial days of studying software development we were introduced to a paradigm called as procedural programming. In this case when the software has to be developed, the entire software is thought of in terms of small procedures or functions that would perform a specific task. These procedures and functions are then taken together, integrated and we get the entire software ready. But the procedural based software turns into something called as “spaghetti-code”. Adding or removing the functionality form the software becomes difficult. Also when the code begins to undergo modification, it becomes difficult to understand and even more difficult to change. This is because all the modules or functions or procedures written for the software are very closely tied to each other. Nothing is independent.

Due to all these limitations in procedural programming there came the paradigm known as object oriented programming language. Here when the software is to be developed, the entire project is viewed in terms of objects and classes. Every class is supposed to have a set of attributes and methods. Here the objects interact with each other to accomplish a purpose. The classes can be reused whenever required. Languages such as Java make use of Object Oriented Programming.

But again object oriented programming has some limitations like code scattering and code tangling. In case of code scattering the same piece of code is spread across the entire software. In case of code tangling, the modules in the software interact simultaneously with many requirements at a time. Thus the software becomes poor in terms of its code traceability. It becomes less productive and does not remain reusable. When the software undergoes change, its size goes on increasing. So it becomes brittle and difficult to change. Due these limitations we have Aspect Oriented Programming Languages. AspectJ is one such language that uses AOP.

Aspect Oriented Programming

Aspect Oriented Programming is used to address the problem of cross cutting concerns. A concern is nothing but a goal or something that we wish to achieve. These concerns are spread across the software. Examples of some cross cutting concerns would be exception handling, logging, security, synchronization etc. In case of object oriented programming these concerns are handled by writing classes. So the code tends to scatter and become entangled. It becomes very difficult to make a slight change to the code. And this is where AOP comes into picture. It introduces us to a new concept called as aspect. An Aspect is a class that encapsulates the cross cutting concern. It also tells how these concerns should be woven into the software. Thus it allows our programs to be more clear, flexible, clean and easy to understand. Here the main job of the programmer is to focus only on the cross cutting concerns.

Aspect Oriented Programming was first invented by the Xerox Palo Alto Research Center (Xerox PARC) in the 20th century. It is implemented by using a variety of tools. Some of the tools that have this functionality are AspectC, an extension to the C language, AspectC++, an aspect oriented extension to C++ language, AspectJ, an aspect oriented extension to the Java language.

AspectJ

AspectJ is an aspect oriented extension created for the Java programming language. It uses the syntax of the Java language. Therefore every valid Java program is also a valid AspectJ program. The AspectJ compiler also produces the class files that conform to the Java byte code specification. Thus any compliant JVM can execute these class files. The AspectJ language consists of two parts

1. The Language Specification and
2. The Language Implementation

The language specification defines the language in which you write the code with AspectJ. The language
implementation part provides the tools for compiling, debugging and integrating with IDE’. AspectJ defines two types of
crosscutting:
1. Static
2. Dynamic

Static crosscutting is nothing but weaving the modification into the static structure like the classes, interfaces and other
aspects of the software. Dynamic crosscutting is nothing but weaving the modification into the program while it is
executing.

Following are the crosscutting elements defined in AspectJ:
1. Joinpoint: It is a point that can be identified in a program. It can be an assignment statement, a call to a method
etc. For eg. Consider the following class
   public class Mobile
   {
   void sendMessage(String message)
   {
   message+="End";
   System.out.println("Message sent");
   }
   }
   In the above example, the joinpoint is the execution of the method sendMessage() and the access to the message
   variable.
2. Pointcut: It is a construct that will select the joinpoint and collect the context at that point. For Eg. We can write a pointcut that will catch the execution of the sendMessage() method in the class Mobile. The pointcut would look like as shown below
   Execution(void Mobile.sendMessage(String))
3. Advice: It is a piece of code that will execute at the joinpoint that has been selected by the pointcut. An advice
can execute either before or after or around the joinpoint. For Eg: The following piece of code shows the before advice
   before(): execution(void Mobile.sendMessage(String))
   {
   System.out.println("On the way to send message");
   }
4. Introduction: It is the static crosscutting instruction that will present change to the classes, interfaces and
   aspects of the software.
5. Aspect: It is the core part of AspectJ. It is the one that will contain the code to represent the weaving rules. The
   pointcuts, advices, joinpoints, introductions are all put together in an aspect. For Eg.
   public aspect LogMessage
   {
   before():sendMessage()
   {
   System.out.println("Executing before sending …");
   }
   }

An aspect is much similar to a class. It can contain data members and methods. It can have access specifications. It can
be declared as abstract. It can extend classes, abstract aspects as well as implement interfaces. Also aspects cannot be
directly instantiated and cannot inherit from concrete aspects.

An Example of Aspect Oriented Programming using AspectJ

Here we present the popular game of TicTacToe written using Java language. The game consists of six classes. Mark,
Player, PlayerTester, TicTacToeBoardTester, TicTacToeGame, TicTacToeBoard.
The source code of the game is presented below
public class Mark {
   X, O;
   @Override
   public String toString()
   {
   if (this == Mark.X) {
   return "X";
   } else {
   return "O";
   }
   }
   public enum Mark {
   X, O;
   @Override
   public String toString() {
   if (this == Mark.X) {
   return "X";
   } else {
   return "O";
   }
   }
Player.java

```java
public class Player {
    String pname;
    int wins=0, ties=0, no_games=0, losses=0;
    Mark pmark;

    public Player(String name) {
        pname = name;
    }

    public String getName() {
        return pname;
    }

    public Mark getMark() {
        return pmark;
    }

    public int getGames() {
        return no_games;
    }

    public int getWins() {
        return wins;
    }

    public int getTies() {
        return (this.ties);
    }

    public int getLosses() {
        return (this.losses);
    }

    public void setMark(Mark mark) {
        this.pmark = mark;
    }

    public void updateScores(TicTacToeBoard board) {
        pmark = board.lastmark;
        this.no_games = this.no_games + 1;
        if (board.wins(pmark))
            this.wins = this.getWins() + 1;
        else
            this.losses = this.getLosses() + 1;
    }

    @Override
    public String toString() {
        String msg;
        return "O";
    }
}
```

Player.java

```java
public class Player {
    String pname;
    int wins=0, ties=0, no_games=0, losses=0;
    Mark pmark;

    public Player(String name) {
        pname = name;
    }

    public String getName() {
        return pname;
    }

    public Mark getMark() {
        return pmark;
    }

    public int getGames() {
        return no_games;
    }

    public int getWins() {
        return wins;
    }

    public int getTies() {
        return (this.ties);
    }

    public int getLosses() {
        return (this.losses);
    }

    public void setMark(Mark mark) {
        this.pmark = mark;
    }

    public void updateScores(TicTacToeBoard board) {
        pmark = board.lastmark;
        this.no_games = this.no_games + 1;
        if (board.wins(pmark))
            this.wins = this.getWins() + 1;
        else
            this.losses = this.getLosses() + 1;
    }

    @Override
    public String toString() {
        String msg;
        return "O";
    }
}
```
msg="Name : "+this.getName()+" Games: "+this.getGames()+" Ties: "+this.getTies()+" Wins: "+this.getWins()+" Losses: "+this.getLosses();

return msg;
}
}

PlayerTester.java

public class PlayerTester{
    public static void main(String []args){
        Player p1 = new Player("joe");
        Player p2 = new Player("amy");
        TicTacToeBoard board = new TicTacToeBoard(3);

        test("getName");
        if(p1.getName().equals("joe")){
            pass();
        } else{
            fail();
        }

        test("setMark to X & getMark");
        p1.setMark(Mark.X);
        if(p1.getMark() == Mark.X){
            pass();
        } else{
            fail();
        }

        test("setMark to O & getMark");
        p2.setMark(Mark.O);
        if(p2.getMark() == Mark.O){
            pass();
        } else{
            fail();
        }

        test("getWins #1");
        if(p1.getWins() == 0){
            pass();
        } else{
            fail();
        }

        board.reset();
        board.play(p1,0,0);
        board.play(p2,1,1);
        board.play(p1,0,2);
        board.play(p2,0,1);
        board.play(p1,2,1);
        board.play(p2,1,0);
        board.play(p1,1,2);
        board.play(p2,2,2);
        board.play(p1,2,0);
        p1.updateScores(board);
        p2.updateScores(board);
        board.reset();
        board.play(p1,0,1);
        board.play(p2,0,0);
board.play(p1,1,1);
board.play(p2,0,2);
board.play(p1,2,1);
p1.updateScores(board);
p2.updateScores(board);
test("getGames");
if(p1.getGames() == 2){
    pass();
} else{
    fail();
}
test("getTies");
if(p1.getTies() == 1){
    pass();
} else{
    fail();
}
test("getLosses");
if(p2.getLosses() == 1){
    pass();
} else{
    fail();
}
}

public static void test(String name){
    System.out.print(name+" test: ");
}

public static void pass(){
    System.out.println("PASS");
}

public static void fail(){
    System.out.println("FAIL");
}

TicTacToeBoard.java

public class TicTacToeBoard {

    int MIN_SIZE=3;
    private static final int ROW = 3;
    private static final int COL = 3;
    private char[][] board ;
    int boardsize;
    int totalmarks;
    Mark lastmark;

    public TicTacToeBoard (int n)
    {
        boardsize=n;
        board = new char[n][n];
        if (n>3)
        {
            StringBuilder str = new StringBuilder("Tic Tac Toe Game board: \n");
            for (int i = 0; i < n; i++)
            {

```
for (int j = 0; j < n; j++) {
    board[i][j] = ' ';
}
str.append("\n");
System.out.println(str.toString());
else
    System.out.println("Please enter a value greater than or equal to three");
}

public void reset() {
    this.boardsize=3;
    for (int i = 0; i < getSize(); i++) {
        for (int j = 0; j < getSize(); j++) {
            board[i][j] = ' ';
        }
    }
    System.out.println("Game reset");
}

public int getSize() {
    return boardsize;
}

public boolean isFull() {
    boolean flag = true;
    for (int i = 0; i < getSize(); i++) {
        for (int j = 0; j < getSize(); j++) {
            if(board[i][j] == ' ')
                flag = false;
        }
    }
    if (flag == true)
        return true;
    else
        return false;
}

public boolean wins( Mark mark ) {
    Mark m = mark;
    if(m==mark.X)
    {
        for(int x = 0;x < getSize();x++) // Rows
            {
            if(board[x][0] == 'X' && board[x][1] == 'X' && board[x][2] == 'X')
return true;
}

for(int y = 0;y < getSize();y++) // Columns
{
    if(board[0][y]=='X' && board[1][y]=='X' && board[2][y]=='X')
        return true;
}
if(board[0][0]=='X' && board[1][1]=='X' && board[2][2]=='X')
    return true;
if(board[2][0]=='X' && board[1][1]=='X' && board[0][2]=='X')
    return true;
}
if(m==Mark.O)
{
    for(int x = 0;x < getSize();x++) // Rows
    {
        if(board[x][0]=='O' && board[x][1]=='O' && board[x][2]=='O')
            return true;
    }
    for(int y = 0;y < getSize();y++) // Columns
    {
        if(board[0][y]=='O' && board[1][y]=='O' && board[2][y]=='O')
            return true;
    }
    if(board[0][0]=='O' && board[1][1]=='O' && board[2][2]=='O')
        return true;
    if(board[2][0]=='O' && board[1][1]=='O' && board[0][2]=='O')
        return true;
}
return false;
}

public boolean isCat() {
    if (isFull() && !wins(Mark.O) && !wins(Mark.X)) {
        System.out.println("Board Cat");
        return true;
    } else {
        System.out.println("Board Not Cat");
        return false;
    }
}

public boolean isGameOver() {
}
// Your code goes here
if(isCat())
    return true;
if (wins(Mark.O) || wins(Mark.X))
{
    System.out.println("Game Over");
    return true;
}
else
{
    System.out.println("Game Not Over");
    return false;
}
}
public boolean play(Player player, int row, int column ) {
    char symbol=' ';
    if (row <= this.getSize() && column < this.getSize())
    {
        Mark m= player.getMark();
        System.out.println(m);
        if (m == Mark.O)
            symbol='O';
        if(m== Mark.X)
            symbol='X';
        if (isX(row,column) || isO(row,column) ||isFull())
            {
                System.out.println("Try another position !!!");
                return false;
            }else
                {board[row][column]=symbol;
                return true;
            }
    }else
        return false;
}
public boolean isX( int row, int column ) {
    if(board[row][column]=='X')
        return true;
    else
        return false;
}
public boolean isO( int row, int column ) {
    if(board[row][column]=='O')
        return true;
    else
        return false;
}
public boolean isVacantSquare( int row, int column ) {
    if(board[row][column]=='#')
        return true;
    else
        return false;
public String toString() {  
int j,i;  
String r = "";  
for( i = 0; i < 3; i++)  
{  
r += " \n";  
for( j = 0; j < 3; j++)  
{  
r += board[i][j];  
r += | ";  
}  
r += " \n";  
if(i==2);  
else  
r+="++++++++++";  
}  
return r;  
}

TicTacToeGame.java
import java.util.Scanner;
public class TicTacToeGame {  
public static void main(String args[])  
{  
Scanner s = new Scanner(System.in);  
int r,c;  
boolean markXWins=false;  
boolean markOWins=false;  
String ans=null;  
System.out.println("Welcome to the game.");  
Player p1= new Player("MARY");  
p1.setMark(Mark.X);  
Player p2= new Player("JOE");  
p2.setMark(Mark.O);  
System.out.println("Player 1’s name : "+p1.getName() +"Mark: "+p1.getMark());  
System.out.println("Player 2’s name : "+p2.getName()+"Mark: "+p2.getMark());  
System.out.println("New Run of the Tic Tac Toe on a 3 * 3 board");  
TicTacToeBoard b1 = new TicTacToeBoard(3);  
do  
{  
while((b1.isCat()!= true) && (b1.isGameOver()!= true) )  
{  
System.out.print(b1.toString());  
do  
{  
System.out.println("Player "+p1.getName()+"’s turn");  
System.out.println("Row :");  
r = s.nextInt();  
System.out.println("Col :");  
c = s.nextInt();  
if(b1.isFull() && (b1.isCat()))  
System.out.println("Board is fullllllllll");  
}  
while(b1.play(p1,r,c)!=true);  
if(b1.wins(Mark.X))  
{
```
System.out.print(p1.getName()+" wins");
System.out.print(b1.toString());
b1.lastmark=Mark.X;
p1.updateScores(b1);
b1.lastmark=Mark.O;
p2.updateScores(b1);
System.out.println(p1.toString());
System.out.println(p2.toString());
markXWins=true;
System.out.println("Do you want to play again?");
ans = s.next();
if(ans.equals("Y")||ans.equals("y"))
    b1.reset();
else
    System.exit(1);
}
System.out.println(b1.toString());

do 
{
    System.out.println("Player "+p2.getName()+"'s turn");
    System.out.println("Row :");
    r = s.nextInt();
    System.out.println("Col :");
    c = s.nextInt();
    if(b1.isFull() && (b1.isCat()))
        System.out.println("Board is fullllllll");
}
while(b1.play(p2,r,c)!=true);
if(b1.wins(Mark.O))
{
    System.out.println(p2.getName()+" wins\n");
    System.out.println(b1.toString());
b1.lastmark=Mark.O;
p1.updateScores(b1);
b1.lastmark=Mark.X;
p2.updateScores(b1);
System.out.println(p2.toString());
System.out.println(p1.toString());
markOWins=true;
System.out.println("Do you want to play again?");
ans = s.next();
if(ans.equals("Y")||ans.equals("y"))
    b1.reset();
else
    System.exit(1);
}
if(markOWins!=true && markXWins!=true)
{
p1.ties=p1.getTies()+1;
p2.ties=p2.getTies()+1;
}
System.out.println("Do you want to play again?");
ans = s.next();
if(ans.equals("Y")||ans.equals("y"))
    b1.reset();
    while(ans.equals("Y") || ans.equals("y"));
}
```
public class TicTacToeBoardTester {
    public static void main(String[] args) {
        TicTacToeBoard board = new TicTacToeBoard(3);
        Player p1 = new Player("Jack");
        Player p2 = new Player("Jill");
        p1.setMark(Mark.X);
        p2.setMark(Mark.O);
        test("getSize");
        if (board.getSize() == 3) {
            pass();
        } else {
            fail();
        }
        test("play #1");
        if (!board.play(p1,10,1))
            pass();
        else
            fail();
        test("play #2");
        if (board.play(p1,0,1))
            pass();
        else
            fail();
        test("play #3");
        if (!board.play(p1,0,2))
            pass();
        else
            fail();
        test("play #4");
        if (board.play(p2,0,2))
            pass();
        else
            fail();
        test("play #5");
        if (!board.play(p1,0,2))
            pass();
        else
            fail();
        test("isX #1");
        if (!board.isX(0,0))
            pass();
        else
            fail();
        test("isX #2");
    }
}
if (board.isX(0,1))
    pass();
else
    fail();

test("isO");
if (board.isO(0,2))
    pass();
else
    fail();

test("isVacantSquare #1");
if (board.isVacantSquare(0,0))
    pass();
else
    fail();

test("isVacantSquare #2");
if (!board.isVacantSquare(0,1))
    pass();
else
    fail();

test("wins #1");

board.reset();
board.play(p1,0,0);
board.play(p2,0,1);
board.play(p1,1,1);
board.play(p2,0,2);
board.play(p1,2,2);
if (board.wins(Mark.X))
    pass();
else
    fail();

test("wins #2");
board.reset();
board.play(p1,2,0);
board.play(p2,0,0);
board.play(p1,1,1);
board.play(p2,2,1);
board.play(p1,0,2);
if (board.wins(Mark.X))
    pass();
else
    fail();

test("wins #3");
board.reset();
board.play(p1,1,0);
board.play(p2,0,0);
board.play(p1,1,1);
board.play(p2,2,2);
board.play(p1,1,2);
if (board.wins(Mark.X))
    pass();
else
    fail();

test("wins #4");
board.reset();
board.play(p1,0,1);
board.play(p2,0,0);
board.play(p1,1,1);
board.play(p2,0,2);
board.play(p1,2,1);

if (board.wins(Mark.X))
    pass();
else
    fail();
test("isFull #1");
board.reset();
if (!board.isFull())
    pass();
else
    fail();

test("isFull #2");
board.reset();
board.play(p1,0,0);
board.play(p2,1,1);
board.play(p1,0,2);
board.play(p2,0,1);
board.play(p1,2,1);
board.play(p2,1,0);
board.play(p1,1,2);
board.play(p2,2,2);
board.play(p1,2,0);

if (board.isFull())
    pass();
else
    fail();
test("isCat #1");
if (board.isCat())
    pass();
else
    fail();
test("isCat #2");
board.reset();
board.play(p1,0,0);
board.play(p2,0,2);
board.play(p1,1,1);
board.play(p2,2,0);
board.play(p1,2,1);
board.play(p2,0,1);
board.play(p1,1,0);
board.play(p2,1,2);
board.play(p1,2,2);
if (!board.isCat())
    pass();
else
    fail();
}

public static void test(String name){
    System.out.print(name+" test: ");
}

public static void pass(){
    System.out.println("PASS");
}

public static void fail(){
}
System.out.println("FAIL");
}

Now the above code is the object oriented implementation of the game. Here we will use AspectJ and build up on crosscutting concern.

Concern 1: The system shall keep track all the methods of the Player class that are invoked and will log in execution for the same.
For this we write an advice that logs the execution of all the methods of the Player class. The advice will look like

    before() : call(* Player.*(..))
    {
        System.out.println("The Game started off. Logging it...");
    }

Concern 2: Any changes that are made on the board shall be logged in.
For this, we write an advice as shown below

    after() : execution (* TicTacToeBoard.*(..))
    {
        System.out.println("Logging in the board changes...");
    }

So now we have an aspect which will take care of these two crosscutting concern. It will look like

    public aspect LoggingAspect {
        before() : call(* Player.*(..))
        {
            System.out.println("The Game started off. Logging it...");
        }
        after() : execution (* TicTacToeBoard.*(..))
        {
            System.out.println("Logging in the board changes...");
        }
    }

Thus we have completed Aspect Oriented Programming using AspectJ. The above project has been compiled and executed on Eclipse

II. CONCLUSIONS

Here we can observe that, if we were to use object oriented programming and log the calls to the method in the Player class or the TicTacToeBoard class, we would have to write the code in individual methods. This will make the system complicated and difficult to manage. But by using aspect oriented programming we are able to achieve the same requirement by simply writing an aspect. This is why Aspect Oriented Programming is used for handling the issue of crosscutting concern.

REFERENCES