Comparative Study of Java and A Non-Imperative Programming Language

Abstract

With the advent of numerous languages including both imperative and non-imperative languages, it is difficult to realize the edge of one language in a particular scope over another one. We are comparing two languages each from one domain and then compare its benefits and specs. We also deeply study which technology is better in which problem environment. Also, we address some core issues of Java and Prolog.

General Term Languages
Keywords, comparing, imperative,non-imperative, core

1. Introduction

1.1. Overview

We will give a brief introduction of the programming languages from an imperative and non-imperative domain. We have selected java from the imperative domain and Prolog from a non-imperative domain. In the next section, we will provide a detailed analysis of Java and nonimperative language prolog with help of a table for better understanding. In the next and second last section, we will discuss which language is better in terms of performance and optimization. All the analysis and inferences are followed by supporting references. The later sections consist of Acknowledgement, Abbreviations, and some compilable code snippets.

1.2. Java

A high-level programming language developed by Sun Microsystems. Java was originally called OAK and was designed for handheld devices and set-top boxes. Oak was unsuccessful so in 1995 Sun changed the name to Java and modified the language to take advantage of the burgeoning World Wide Web. Java is an object-oriented language similar to C++ but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a just-in-time compiler (JIT). Java is a general-purpose programming language with several features that make the language well suited for use on the

World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.

1.3. Prolog

Prolog is a logic programming language. It has an important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called Facts and Rules). The core heart of the prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations. The language was developed and implemented in Marseille, France, in 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses.

Prolog was one of the first logic programming languages and remains the most popular such language today, with several free and commercial implementations available. The language has been used for theorem proving, expert systems, term rewriting, type systems, and automated planning, as well as its original intended field of use, natural language processing. Modern Prolog environments support the creation of graphical user interfaces, as well as administrative and networked applications.

Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates. PROLOG an engine that is entirely written in JAVA has been integrated into JAVA programs using annotations and generics. Rules and facts are written directly into the JAVA code within annotations. The syntax for rules and facts is native PROLOG. This leads to a complex mixture of not only code in different programming languages but different programming paradigms, too.

2. Analysis

2.1. Java Vs Prolog

✓ ***Default more secure programming practices***

- Java provides more secure programming practices as compared to Prolog.

✓ ***Web Application Development***

- Prolog is extremely well suited for developing web applications. Web pages are naturally represented as Prolog terms and can be easily created, inspected, and processed recursively.

- ✓ ***Functional Programming***
    - Java has no functions; however, using interfaces and inner classes it is possible to mimic some but not all the features of functional programming. But Prolog is a functional programming language. It lacks the basic functionalities of declarative programming. It has also built-in functions for arithmetic operations. The prolog functional programming styles meshes with both multithreading and transactional-based system.

- ✓ ***Declarative Programming***
    - You can embed some of the declarative programming features in Java using libraries like JSetL and JSolver. Java is far better than Prolog in declarative programming which lacks in prolog. In prolog, we only implement and visualize the solution rather than implement it. Java is pure declarative programming in terms of functions, classes, and interfaces.

- ✓ ***OO-based abstraction***
    - In java, there is more OO-based abstraction as compared to Prolog. In java, we define methods, access modifiers which helps programmer to relate easily.
    - Prolog is a redundant language in the sense there is no OO-based abstraction present in it. The code written in prolog is not readable as compare to java.
    - Lines of code in Prolog are relatively less than java but java code is more reliable and easy to understand to the user.

- ✓ ***UI Prototype Design***
    - java code is more readable in terms of logic writing. The syntax is consistent and easy to understand. In prolog, however, lines of code are relatively less in size but difficult to understand.
    - Indentation and UI design of Java code are far better than prolog which is just simple black and white notepad type code editor.

3. Which one is better? Java or Prolog

Each programming language has its perks and benefits. It also depends on specifically what they like more about the programming language. The analysis provided above in the report in a detailed manner shows that Java and prolog have their benefits and drawbacks side by side but Java has more advantages over prolog in terms of OO-based abstraction, security, and UI prototype design.

Java offers more security than any other language. It has an automated garbage collector. Buffer overflow exceptions are some other security features provided by java. Moreover, java has no concept of the pointer.

4. Conclusion

After carefully analyzing the languages, we have concluded that every language has its ups and downs. Every particular language has a purpose but can be extended or revised to accommodate the current needs of programming. Despite all this, every language has its specialty and considerably better programming practices which have made it popular and revolutionized the computing world. Java is now the dominant language because of its unique features and multiples software compatibility. A major drawback of prolog is that it has compatibility issues with software which is of major type.

5. Appendix

5.1. Java Code for Chess

Piece.java

```
package com.company;

import java.util.LinkedList;

public class Piece {
   int xp;
   int yp;
   boolean isWhite;
   LinkedList<Piece> ps;
   String name;

   public Piece(int xp, int yp, boolean isWhite,String n, LinkedList<Piece> ps) {
      this.xp = xp;
      this.yp = yp;
      this.isWhite = isWhite;
      this.ps=ps;
      name=n;
      ps.add(this);
   }


}
```

Main.java

```
public class Main {

   public static int Empty = 0;
   public static int King = 1;
   public static int Queen = 2;
   public static int Bishop = 3;
   public static int Knight = 4;
   public static int Rook = 5;

   static public int[][] board = new int[8][8];


   public static void main(String[] args) throws IOException {
      // write your code here
      LinkedList<Piece> ps=new LinkedList<>();
      BufferedImage all= ImageIO.read(new File("D:\\chess.png"));
      Image []imgs=new Image[12];
      int ind=0;
      int []BackRow=new int[8];

      //Parse the images from the pic and store in the List
      for(int y=0;y<400;y+=200){
         for(int x=0;x<1200;x+=200){
            imgs[ind]=all.getSubimage(x,  y,  200,  200).getScaledInstance(64,  64,
BufferedImage.SCALE_SMOOTH);
            ind++;
         }
      }

     //Generate Indices of the chess according to mentioned rules
      BackRow=Random_Chess_Board(board, 8);

      //store the piece(Kind,queen etc) in form of linked list
      Chess_Placement_Block(ps,BackRow);

      //create black and white boxes in the frame
      JFrame frame = new JFrame();
      frame.setBounds(10, 10, 512, 512);
      frame.setUndecorated(true);
      JPanel pn = new JPanel() {
         @Override
```

```
    public void paint(Graphics g) {
        boolean white = true;
        for (int y = 0; y < 8; y++) {
            for (int x = 0; x < 8; x++) {
                if (white) {
                    g.setColor(new Color(235, 235, 208));
                } else {
                    g.setColor(new Color(119, 148, 85));

                }
                g.fillRect(x * 64, y * 64, 64, 64);
                white = !white;
            }
            white = !white;
        }
        for(Piece p: ps){
            int ind=0;
            if(p.name.equalsIgnoreCase("king")){
                ind=0;
            }
            if(p.name.equalsIgnoreCase("queen")){
                ind=1;
            }
            if(p.name.equalsIgnoreCase("bishop")){
                ind=2;
            }
            if(p.name.equalsIgnoreCase("knight")){
                ind=3;
            }
            if(p.name.equalsIgnoreCase("rook")){
                ind=4;
            }
            if(p.name.equalsIgnoreCase("pawn")){
                ind=5;
            }
            if(!p.isWhite){
                ind+=6;
            }
            g.drawImage(imgs[ind], p.xp*64, p.yp*64, this);
        }
    }
};
```

```
    frame.add(pn);
    frame.setVisible(true);

}

//print the chess board
static void Print_Board(int[][] board, int size) {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (j % 2 == 0) {
                System.out.print("'" + board[i][j] + "'" + " | ");
            } else {
                System.out.print(board[i][j] + " | ");
            }
        }
        System.out.println();
    }

}

//generate indices of the chess
static int[] Random_Chess_Board(int[][] board, int size) {

    int[] BackRow = new int[8];

    List<Integer> EmptyOdd = new ArrayList<Integer>();
    List<Integer> EmptyEven = new ArrayList<Integer>();

    for (int i = 0; i < 8; i++) {
        if (i % 2 == 0) {
            EmptyEven.add(i);
        } else {
            EmptyOdd.add(i);
        }
    }
    System.out.println("Even List");
    for(int i=0;i<EmptyEven.size();i++){
        System.out.println(EmptyEven.get(i));
    }
    System.out.println("Odd List");
    for(int i=0;i<EmptyOdd.size();i++){
        System.out.println(EmptyOdd.get(i));
    }
```

```
Random rand = new Random();

//Place King in array at index 1 and 6
int min = 1, max = 6;
int KingIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
BackRow[KingIndex] = King;
System.out.println("King Index = " + KingIndex);

//place rooks at random spaces higher and lower than kings place
min = 0;
max = KingIndex - 1;
int rookOneIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
min = KingIndex + 1;
max = 7;
int rookTwoIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
BackRow[rookOneIndex] = Rook;
BackRow[rookTwoIndex] = Rook;
System.out.println("rookOneIndex = " + rookOneIndex);
System.out.println("rookTwoIndex = " + rookTwoIndex);


//Update the odd and even lists so no pieces will be placed on top of the already placed pieces
EmptyEven.remove(new Integer(KingIndex));
EmptyOdd.remove(new Integer(KingIndex));
EmptyEven.remove(new Integer(rookOneIndex));
EmptyOdd.remove(new Integer(rookOneIndex));
EmptyEven.remove(new Integer(rookTwoIndex));
EmptyOdd.remove(new Integer(rookTwoIndex));

//place bishops,1 on odd index,1 on even
min = 0;
max = EmptyEven.size() - 1;
int bishopOneIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
min = 0;
max = EmptyOdd.size() - 1;
int bishopTwoIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
BackRow[EmptyEven.get(bishopOneIndex)] = Bishop;
BackRow[EmptyOdd.get(bishopTwoIndex)] = Bishop;

System.out.println("biShopOneIndex = " + EmptyEven.get(bishopOneIndex));
```

```
        System.out.println("biShopTwoIndex = " + EmptyOdd.get(bishopTwoIndex));

        //Update the odd and even lists so no pieces will be placed on top of the already placed pieces
        EmptyEven.remove(new Integer(EmptyEven.get(bishopOneIndex)));
        EmptyOdd.remove(new Integer(EmptyOdd.get(bishopTwoIndex)));

        //Since the Bishops are placed, only 1 list of empty spaces is needed
        List<Integer> emptySpaces = new ArrayList<Integer>();
        emptySpaces.addAll(EmptyEven);
        emptySpaces.addAll(EmptyOdd);

        System.out.println("Even List");
        for(int i=0;i<EmptyEven.size();i++){
           System.out.println(EmptyEven.get(i));
        }
        System.out.println("Odd List");
        for(int i=0;i<EmptyOdd.size();i++){
           System.out.println(EmptyOdd.get(i));
        }
        System.out.println("Empty Spaces List");
        for(int i=0;i<emptySpaces.size();i++){
           System.out.println(emptySpaces.get(i));
        }

        //Place queen
        min = 0;
        max = emptySpaces.size()-1;
        int queenIndex = (int) Math.floor(Math.random() * (max - min + 1) + min);
        BackRow[emptySpaces.get(queenIndex)] = Queen;

        System.out.println("QueenIndex = " + emptySpaces.get(queenIndex));

        //Update the empty spaces list so no pieces will be placed on top of the already placed pieces
        emptySpaces.remove(new Integer(emptySpaces.get(queenIndex)));

        //Place knights
        int knightOneIndex = emptySpaces.get(0);
        int knightTwoIndex = emptySpaces.get(1);
        BackRow[knightOneIndex] = Knight;
        BackRow[knightTwoIndex] = Knight;
```

```
    System.out.println("KnightOneIndex = " + knightOneIndex);
    System.out.println("KnightTwoIndex = " + knightTwoIndex);

    //Return Back row contains all random indices of the chess


  return BackRow;
}

//place the pieces of chess in the linked list for later use
static void Chess_Placement_Block(LinkedList<Piece> ps,int []BackRow){
    //Row 1
    for(int i=0;i<8;i++){
        System.out.println(BackRow[i]);
        if(BackRow[i]==1){
            Piece bking=new Piece(i, 0, false, "king", ps);
        }
        else if(BackRow[i]==2){
            Piece bqueen=new Piece(i, 0, false, "queen", ps);
        }
        else if(BackRow[i]==3){
            Piece bbishop=new Piece(i, 0, false, "bishop", ps);
        }
        else if(BackRow[i]==4){
            Piece bkinght=new Piece(i, 0, false, "knight", ps);
        }
        else if(BackRow[i]==5){
            Piece brook=new Piece(i, 0, false, "rook", ps);
        }

    }

    //Row 7
    for(int i=0;i<8;i++){
        System.out.println(BackRow[i]);
        if(BackRow[i]==1){
            Piece wking=new Piece(i, 7, false, "king", ps);
        }
        else if(BackRow[i]==2){
            Piece wqueen=new Piece(i, 7, false, "queen", ps);
        }
        else if(BackRow[i]==3){
            Piece wbishop=new Piece(i, 7, false, "bishop", ps);
```

```
        }
        else if(BackRow[i]==4){
           Piece wkinght=new Piece(i, 7, false, "knight", ps);
        }
        else if(BackRow[i]==5){
           Piece wrook=new Piece(i, 7, false, "rook", ps);
        }

     }


     Piece bpawn1=new Piece(1, 1, false, "pawn", ps);
     Piece bpawn2=new Piece(2, 1, false, "pawn", ps);
     Piece bpawn3=new Piece(3, 1, false, "pawn", ps);
     Piece bpawn4=new Piece(4, 1, false, "pawn", ps);
     Piece bpawn5=new Piece(5, 1, false, "pawn", ps);
     Piece bpawn6=new Piece(6, 1, false, "pawn", ps);
     Piece bpawn7=new Piece(7, 1, false, "pawn", ps);
     Piece bpawn8=new Piece(0, 1, false, "pawn", ps);

     Piece wpawn1=new Piece(1, 6, true, "pawn", ps);
     Piece wpawn2=new Piece(2, 6, true, "pawn", ps);
     Piece wpawn3=new Piece(3, 6, true, "pawn", ps);
     Piece wpawn4=new Piece(4, 6, true, "pawn", ps);
     Piece wpawn5=new Piece(5, 6, true, "pawn", ps);
     Piece wpawn6=new Piece(6, 6, true, "pawn", ps);
     Piece wpawn7=new Piece(7, 6, true, "pawn", ps);
     Piece wpawn8=new Piece(0, 6, true, "pawn", ps);

  }



}
```

5.2. Explanation:

- Initially, we kept two arrays of even and odd dices to store white and black.
- Then we make a random chess function which will randomly place the pieces on the board.
- Firstly we generate a random number between 1 and 6 for king and place it into the row and remove the entry from the respective list.

- Then we generate an index to place rooks. One on the left side of the king and the second on the right side of the king. Also, remove it from the respective list.
- Then we place randomly 1 bishop on the odd index and 1 bishop on the even index and remove its entry from the respective list.
- Then we generate a random index of the queen from the space list and place it into the position on the board.
- The remaining index in the list represents the location of two knights on the board.

5.3. Prolog Code for Chess

Chess.pl

```
random(L, U, R):-
  integer(L), integer(U),
  !,
  R is L+random(U-L).

list_delete(R, [Y|L2], [Y|L1]) :-
  list_delete(R,L2,L1).

len([X|Y], LenResult):-
   len(Y, L),
   LenResult is L + 1.

concat([E|L1],L2,[E|C]):-
      con(L1,L2,C).

generateBoard(Board):-
       emptyEven=[2,4,6,8],
       emptyOdd=[1,3,5,7].
       random(2,7,king).
       list_delete(king,emptyEven).
       list_delete(king,emptyOdd).
       random(1,R-1,rook1).
       random(R+1,8,rook2).
       list_delete(rook1,emptyEven).
       list_delete(rook2,emptyEven).
       list_delete(rook1,emptyOdd).
       list_delete(rook2,emptyOdd).
       len(emptyEven,lenB1).
       random(1,lenB1,bishop1).
         len(emptyOdd,lenB2).
         random(1,lenB2,bishop2).
         list_delete(bishop1,emptyEven).
         list_delete(bishop2,emptyOdd).
         concat(emptyEven,emptyOdd,allList).
         len(allList,A).
         random(1,A,queen1).
         list_delete(allList,queen1).
         allList=[X|[Y|T]],
         knight1 = X,
```

```
        knight2 = Y,



    initial(piece(white, rook,   1, 1)).
    initial(piece(white, knight, 2, 1)).
    initial(piece(white, bishop, 3, 1)).
    initial(piece(white, queen,  4, 1)).
    initial(piece(white, king,   5, 1)).
    initial(piece(white, bishop, 6, 1)).
    initial(piece(white, knight, 7, 1)).
    initial(piece(white, rook,   8, 1)).
    initial(piece(white, pawn,   X, 2)) :-
      between(1, 8, X).

    initial(piece(black, rook,   1, 8)).
    initial(piece(black, knight, 2, 8)).
    initial(piece(black, bishop, 3, 8)).
    initial(piece(black, queen,  4, 8)).
    initial(piece(black, king,   5, 8)).
    initial(piece(black, bishop, 6, 8)).
    initial(piece(black, knight, 7, 8)).
    initial(piece(black, rook,   8, 8)).
    initial(piece(black, pawn,   X, 7)) :-
      between(1, 8, X).


initial_board(Board) :-
  findall(Piece, initial(Piece), Board).
```