# *Inheritance Dumb Chess*

A mention is made in this video (near the end) about the chess program being a good candidate for the last assignment in cps161 where you get to create your own project. That idea was later eliminated because it appeared that the amount of work already required in the course was enough and it seemed excessive to add on another project.

1. We are going to start out with a Generic Chess board with generic pieces that move anywhere.
2. Our pieces can capture other pieces when they move onto a square already occupied by another piece.
3. The generic chess board will be represented by:
   - ChessPiece.java (contains code for generic chess pieces)
   - GenericBoard.java (Places generic pieces that start out in the normal places where chess pieces normally reside)
4. Then we will use Inheritance to create something closer to chess (still with lots of things that could be improved)
   - We will create a family of classes that extend ChessPiece (Rook, Bishop, Pawn, Queen, King, Knight). These class show up in SpecificChessPieces.java
   - We will create ChessBoard which extends GenericBoard to place non-generic (i.e. Rook, Knight, etc) onto a ChessBoard and run the game again.
5. Some of the things our Dumb Chess game doesn't do properly (but could be added)
   - No notion of taking turns between white and black pieces
   - the current version allows white pieces to capture white pieces
   - Pawns don't have their normal capture moves on diagonals
   - Doesn't undertand being blocked (this would be much more challenging to implement). Currently all pieces move like Knights that can't be blocked
   - No concept of King checks, castling, etc. (again more challenging)

### ChessPiece.java

```
package chess;

enum ColorType {W, B};

public class ChessPiece
{
    private int row,  col;
    private ColorType colorType;

    ChessPiece(ColorType color, int r, int c)
    {
        colorType = color;
        row = r;
        col = c;
    }
    public String toString()
    {
        ColorType color=getColor();
        return color.toString() + " row="+row + " col="+col;
    }
    ColorType getColor()
    {
        return colorType;
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"X";
    }
    boolean isCaptured()
    {
        if (row < 0)
            return true;
        else
            return false;
    }
    void setCaptured()
    {
        row = -1;
    }
    boolean isAt(int r, int c)
    {
        if (row==r && col==c)
            return true;
        else
```

```
            return false;
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (rowDelta == 0 && colDelta==0)
            return false;

        int newRow = row + rowDelta;
        int newCol = col + colDelta;
        if (newRow >= 0 && newCol >= 0 &&
                newRow < GenericBoard.NUM_ROWS && newCol < GenericBoard.NUM_COLS)
            return true;
        else
            return false;
    }
    boolean move(int rowDelta, int colDelta)
    {
        if (isMoveLegal(rowDelta, colDelta))
        {
            row += rowDelta;
            col += colDelta;
            return true;
        }
        else
            return false;
    }
}
```

**GenericBoard.java**

```
package chess;
import java.util.Scanner;

public class GenericBoard {
        static final int NUM_ROWS=8;
        static final int NUM_COLS=8;
        static final int NUM_CHESS_PIECES=32;

        ChessPiece[] chessPieces = new ChessPiece[NUM_CHESS_PIECES];

        GenericBoard()
        {
                addPiecesToBoard();
        }
        void addPiecesToBoard()
        {
                int index=0;

                // First 2 rows get Black Pieces
                for (int row=0; row < 2; row++)
                {
                        for (int col=0; col < NUM_COLS; col++)
                                chessPieces[index++]= new ChessPiece(ColorType.B, row, col);
                }


                // last 2 rows get White Pieces
                for (int row=6; row < NUM_ROWS; row++)
                {
                        for (int col=0; col < NUM_COLS; col++)
                                chessPieces[index++]= new ChessPiece(ColorType.W, row, col);
                }
        }


        int findPieceAt(int row, int col)
        {

                for (int i=0; i < NUM_CHESS_PIECES; i++)
                {
                        if (chessPieces[i].isAt(row,col))
                        {
                                return i;
                        }
                }
                return -1;
        }
        // Draw the Chess Board
        void displayBoard()
        {
```

```
		System.out.println();
		System.out.println("*****************************************");
		System.out.print("  ");
		for (int c=0; c < NUM_COLS; c++)
			System.out.printf("%4d", c);

		System.out.println();

		for (int r = 0 ; r < NUM_ROWS; r++)
		{
			System.out.println("   ---------------------------------");
			System.out.printf("%d: |", r);
			for (int c=0; c < NUM_COLS; c++)
			{
				int index = findPieceAt(r,c);
				if (index < 0 )
					System.out.print("   |");  // No piece here
				else
					System.out.printf("%3s|", chessPieces[index].getId());  // Label Square wit
			}
			System.out.println();
		}
		System.out.println("   ---------------------------------");

		System.out.print("Captured Pieces: ");
		for (int i=0; i < NUM_CHESS_PIECES; i++)
		{
			if (chessPieces[i].isCaptured())
				System.out.printf("%3s", chessPieces[i].getId());
		}
		System.out.println();
		System.out.println("+++++++++++++++++++++++++++++++++++++++++++++");
	}
	boolean move(int curRow, int curCol, int nextRow, int nextCol)
	{
		int pieceIndex = findPieceAt(curRow,  curCol);
		if (pieceIndex < 0)
		{
			System.out.println("No piece at specified location");
			return false;
		}
		ChessPiece chessPiece = chessPieces[pieceIndex];

		int otherPieceIndex = findPieceAt(nextRow, nextCol);
		if (chessPiece.move(nextRow-curRow, nextCol-curCol))
		{
			if (otherPieceIndex >= 0)
			{
				chessPieces[otherPieceIndex].setCaptured();
			}
			return true;
		}
		else
		{
			System.out.println("Illegal move for: "+ chessPiece.toString());
			return false;
		}
	}
	void gameLoop()
	{
		boolean continueRunning = true;
		Scanner keyboard = new Scanner(System.in);

		displayBoard();

		while (continueRunning)
		{
			System.out.println("Enter curRow, curCol, nextRow, nextCol");
			int curRow = keyboard.nextInt();
			int curCol = keyboard.nextInt();
			int nextRow = keyboard.nextInt();
			int nextCol = keyboard.nextInt();

			if (move(curRow, curCol, nextRow, nextCol))
			{
				displayBoard();
			}
			else
				System.out.println("Bad move choice ... try again bozo ");

		}
```

```
        }

        public static void main(String[] args)
        {
                GenericBoard genericBoard = new GenericBoard();
                genericBoard.gameLoop();
        }


}
```

**SpecificChessPieces.java**

```
package chess;

class Rook extends ChessPiece
{
    Rook(ColorType color, int r, int c)
    {
        super(color, r, c);
    }
    public String toString()
    {
        return "Rook: " + super.toString();
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"R";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            if (rowDelta==0 || colDelta==0)
                return true;
        }
        return false;
    }
}

class Bishop extends ChessPiece
{
    Bishop(ColorType color, int r, int c)
    {
        super(color, r, c);
    }
    public String toString()
    {
        return "Bishop: " + super.toString();
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"B";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            if (Math.abs(rowDelta) == Math.abs(colDelta))
                return true;
        }
        return false;
    }
}

class Knight extends ChessPiece
{
    Knight(ColorType color, int r, int c)
    {
        super(color, r, c);
    }

    public String toString()
    {
        return "Knight: " + super.toString();
    }
    String getId()
    {
```

```
        ColorType color = getColor();
        return color.toString()+"N";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            rowDelta = Math.abs(rowDelta);
            colDelta = Math.abs(colDelta);
            if ((rowDelta==1 && colDelta==2) ||
                    (rowDelta==2 && colDelta==1))
                return true;
        }
        return false;
    }
}


class Queen extends ChessPiece
{
    Queen(ColorType color, int r, int c)
    {
        super(color, r, c);
    }

    public String toString()
    {
        return "Queen: " + super.toString();
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"Q";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            if (Math.abs(rowDelta) == Math.abs(colDelta))
                return true;
            if (rowDelta == 0 || colDelta == 0)
                return true;
        }
        return false;
    }
}

class King extends ChessPiece
{
    King(ColorType color, int r, int c)
    {
        super(color, r, c);
    }

    public String toString()
    {
        return "King: " + super.toString();
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"K";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            if ((rowDelta >= -1 && rowDelta <=1) &&
                    (colDelta >= -1 && colDelta <=1))
                return true;
        }
        return false;
    }
}

class Pawn extends ChessPiece
{
    Pawn(ColorType color, int r, int c)
    {
        super(color, r, c);
    }
```

```
    public String toString()
    {
        return "Pawn: " + super.toString();
    }
    String getId()
    {
        ColorType color = getColor();
        return color.toString()+"P";
    }
    boolean isMoveLegal(int rowDelta, int colDelta)
    {
        if (super.isMoveLegal(rowDelta, colDelta))
        {
            if (colDelta != 0)
                return false;
            ColorType color = getColor();
            if ( color == ColorType.B && (rowDelta == 1 || rowDelta==2))
                return true;
            if ( color == ColorType.W && (rowDelta==-1 || rowDelta==-2))
                return true;
        }
        return false;
    }
}
```

**ChessBoard.java**

```
package chess;

public class ChessBoard extends GenericBoard{

    int index=0;

    void addKingRowPiece(ColorType c, int row, int col)
    {
        switch (col)
        {
        case 0:
        case 7:
            chessPieces[index++]= new Rook(c, row, col);
            break;
        case 1:
        case 6:
            chessPieces[index++] = new Knight(c,row, col);
            break;
        case 2:
        case 5:
            chessPieces[index++] = new Bishop(c,row, col);
            break;
        case 3:
            chessPieces[index++] = new Queen(c,row, col);
            break;
        case 4:
            chessPieces[index++] = new King(c,row, col);
            break;
        }
    }

    void addPiecesToBoard()
    {
        // Black King Row
        for (int col=0; col < NUM_COLS; col++)
            addKingRowPiece(ColorType.B, 0,  col);

        // Black Pawn Row
        for (int col=0; col < NUM_COLS; col++)
            chessPieces[index++] = new Pawn(ColorType.B, 1, col);

        // White Pawn Row
        for (int col=0; col < NUM_COLS; col++)
            chessPieces[index++] = new Pawn(ColorType.W, 6, col);

        // White King Row
        for (int col=0; col < NUM_COLS; col++)
            addKingRowPiece(ColorType.W, 7,  col);
    }

    public static void main(String[] args) {
        ChessBoard chessBoard = new ChessBoard();
```

```
        chessBoard.gameLoop();

    }

}
```

---

Last Modified: January 6, 2014 10:27 PM