```java
package main;

/*
 * Author: Jon Wyrick
 *
 * The program runs in the directory where vasp data has been calculated and a WAVECAR file
 * has been generated.  It reads in the WAVECAR file, modifies the occupations of KS states
 * so that they are occupied only within some specified KS energy range, and then outputs
 * a new WAVECAR file with the specified occupations as WAVECAR.out.
 *
 */

//import necessary io functionality
import java.io.*;
import java.nio.*;

public class ProjectWave
{
	public static void main(String[] args)
	{
		double eMin = 0;  //lower end of energy range
		double eMax = 0;  //upper end of energy range
		double occupation = 0;  //occupation to be given to orbitals in energy range

		try
		{
			//set the energy range and preferred occupation from the command line inputs
			eMin = Double.valueOf(args[0]);
			eMax = Double.valueOf(args[1]);
			occupation = Double.valueOf(args[2]);

			//swap min and max if they were input backwards
			double tmp = eMax;
			if (eMin > eMax)
			{
				eMax = eMin;
				eMin = tmp;
			}
		}
		catch (Exception ex)
		{
			//print argument specification if wrong parameters were input, and then exit
			System.out.println("Specify energy range in arguements: E-min E-max, and
			preferred occupation within energy range");
			System.exit(0);
		}

		//the file to be read in will be WAVECAR, and the file to be out put will be called
		WAVECAR.out
		File f = new File("WAVECAR");
		File fileOut = new File("WAVECAR.out");

		try
		{
			//set up input and output streams for the 2 files respectively
			FileInputStream in = new FileInputStream(f);
			FileOutputStream out = new FileOutputStream(fileOut);

			try
			{
				//WAVECAR is binary with a little endian byte order, so bytes
				//will be read in using the byte buffer bIn, which are interpreted
				//as doubles via the double buffer dIn:
				ByteBuffer bIn = ByteBuffer.allocate(24);//fortran assumes an initial
				chunk of data of 24 bytes
				bIn.order(ByteOrder.LITTLE_ENDIAN);
				DoubleBuffer dIn = bIn.asDoubleBuffer();

				//rec is the array that stores the doubles read in from the WAVECAR
```

```
67                              //((which is a binary file of fortran records).
68                              double[] rec;
69
70                              //the readRecord function is defined later, but for convenience the
71                              //meanings of the parameters it takes are defined here:
72                              //
73                              //readRecord(
74                              //      number of bytes per record,
75                              //      number of doubles containing data in record,
76                              //      file input stream,
77                              //      byte input stream,
78                              //      double view of byte input stream )
79
80                              //the first record contains 3 doubles (24 bytes):
81                              //  recordLength redefines the record length to be something much
                                larger than 24 bytes
82                              //  numSpin is the number of spins
83                              //  precisionFlag determines what precision data values have
84                              rec = readRecord(24, 3, in, bIn, dIn);
85
86                              int recordLength = (int)rec[0];
87                              int numSpin = (int)rec[1];
88                              int precisionFlag = (int)rec[2];
89
90                              //print out the information read in so far
91                              System.out.println("record length: \t" + recordLength);
92                              System.out.println("num spin: \t" + numSpin);
93                              System.out.println("prec flag: \t" + precisionFlag);
94                              System.out.println();
95
96                              //with the record length redefined, we need to re-read in the file,
97                              //and then go to the second record.
98                              ByteBuffer bIn2 = ByteBuffer.allocate(recordLength);
99                              bIn2.order(ByteOrder.LITTLE_ENDIAN);
100                             DoubleBuffer dIn2 = bIn2.asDoubleBuffer();
101                             FloatBuffer fIn2 = bIn2.asFloatBuffer();
102
103                             in.close();
104
105                             //for the output file, we copy the header, so the first record should
                                be written as is
106                             ByteBuffer bOut = ByteBuffer.allocate(recordLength);
107                             bOut.order(ByteOrder.LITTLE_ENDIAN);
108                             DoubleBuffer dOut = bOut.asDoubleBuffer();
109                             FloatBuffer fOut = bOut.asFloatBuffer(); //floating point output to be
                                used later
110                             //some data is stored as doubles, and some as floats.
111
112                             //writeRecord( bytes in the record,
113                             //             record data,
114                             //             file output stream,
115                             //             byte output stream,
116                             //             double output stream )
117                             writeRecord(recordLength,rec,out,bOut,dOut);
118
119                             //re-open the file
120                             in = new FileInputStream(f);
121
122                             //skip the first record since we already used it by performing a read
                                and not recording it.
123                             readRecord(recordLength, 0, in, bIn2, dIn2);
124
125                             //now read in information on the number of k-points per band, the
                                number of bands,
126                             //and the energy cutoff.
127                             rec = readRecord(recordLength, 12, in, bIn2, dIn2);
128                             int numKPoints = (int)rec[0];
129                             int numBands = (int)rec[1];
130                             double energyCutoff = rec[2];
```

```
131
132                             //we can also read in the super cell, as is done here, though it's not
                                necessary:
133                             double[][] a = new double[3][3];
134                             for (int i = 0; i < 3; i ++)
135                             {
136                                     for (int j = 0; j < 3; j ++)
137                                             a[i][j] = rec[3*i + j + 3];
138                             }
139
140                             System.out.println("num k-points: \t" + numKPoints);
141                             System.out.println("num bands: \t" + numBands);
142                             System.out.println("energy cutoff: \t" + energyCutoff);
143
144                             System.out.println();
145                             for (int i = 0; i < 3; i ++)
146                                     System.out.println("a" + i + ":  \t" + a[i][0] + "\t" + a[i][1]
                                        + "\t" + a[i][2]);
147
148
149                             System.out.println();
150                             System.out.println();
151
152                             //copy this record to the output file
153                             writeRecord(recordLength,rec,out,bOut,dOut);
154
155                             //loop through each k-point, getting information about each band at
                                each k-point
156                             for (int kIdx = 0; kIdx < numKPoints; kIdx ++)
157                             {
158                                     rec = readRecord(recordLength, 4 + numBands*3, in, bIn2, dIn2);
159                                     int numPlaneWaves = (int)rec[0];//number of plane waves used as
                                        a basis for each band
160                                     double[] k = new double[]{rec[1],rec[2],rec[3]};//k-space
                                        vector of this k-point
161
162                                     //print the results for the first 4 k-points to see that
                                        everything is working
163                                     if (kIdx < 4)
164                                     {
165                                             System.out.println("num plane waves: " +
                                                numPlaneWaves);
166                                             System.out.println("k: \t" + k[0] + "\t" + k[1] + "\t"
                                                + k[2]);
167                                     }
168
169                                     //KS energy is treated as a complex number, though it only has
                                        a real part
170                                     //there will be one KS energy for each band.
171                                     double[] realEnergy = new double[numBands];
172                                     double[] imaginaryEnergy = new double[numBands];
173
174                                     //bandOccupation is the occupation of any given band
175                                     double[] bandOccupation = new double[numBands];
176                                     int idx = 4;//the first 4 values were already read in above as:
                                        numPlaneWaves and the vector k
177
178                                     //loop through each band at this k-point
179                                     for (int i = 0; i < numBands; i ++)
180                                     {
181                                             realEnergy[i] = rec[idx++];
182                                             imaginaryEnergy[i] = rec[idx++];
183
184                                             bandOccupation[i] = rec[idx];
185
186                                             //set bands inside of energy range to be occupied and
                                                bands outside range to be unoccupied
187                                             if ((realEnergy[i] >= eMin) && (realEnergy[i] <= eMax))
188                                                     rec[idx] = occupation;
```

```
189                                             else
190                                                     rec[idx] = 0;
191
192                                             idx ++;
193                                     }
194
195                                     //for the first 4 k-points we also print out info about 4 bands
                                        from each,
196                                     //again as a check that the data is being properly read.
197                                     if (kIdx < 4)
198                                     {
199                                             for (int i = 0; i < 4; i ++)
200                                             {
201                                                     System.out.println("Band Energy " + i + ": \t"
                                                        + realEnergy[i] + " + " + imaginaryEnergy[i] +
                                                        "i");
202                                                     System.out.println("Band Occupation: " +
                                                        bandOccupation[i]);
203                                             }
204                                     }
205
206                                     //having only modified the occupations, but otherwise left
                                        everything the
207                                     //same, we can now write the record to file.
208                                     writeRecord(recordLength,rec,out,bOut,dOut);
209
210                                     //the next set of records contain the plane wave coefficients.
211                                     //the plane wave coefficients are stored as complex floats (one
                                        float for the real
212                                     //part and one float for the imaginary part).
213                                     float[][] realCoefficients = new float[numBands]
                                        [numPlaneWaves];
214                                     float[][] imaginaryCoefficients = new float[numBands]
                                        [numPlaneWaves];
215
216                                     //loop through each band and get the plane wave coefficients
217                                     for (int bandIdx = 0; bandIdx < numBands; bandIdx ++)
218                                     {
219                                             float[] recF = readRecordF(recordLength,
                                                numPlaneWaves*2, in, bIn2, fIn2);
220                                             int idxF = 0;
221                                             for (int i = 0; i < numPlaneWaves; i ++)
222                                             {
223                                                     realCoefficients[bandIdx][i] = recF[idxF++];
224                                                     imaginaryCoefficients[bandIdx][i] =
                                                        recF[idxF++];
225                                             }
226
227                                             //then write the record of coefficients unchanged.
228                                             writeRecordF(recordLength,recF,out,bOut,fOut);
229                                     }
230
231                                     //print out example coefficients for the 1st 4 k-points to see
                                        that everything
232                                     //read fine.
233                                     if (kIdx < 4)
234                                     {
235                                             System.out.println();
236                                             for (int i = 0; i < 4; i ++)
237                                             {
238                                                     System.out.println("Band " + i);
239                                                     for (int j = 0; j < 4; j ++)
240                                                             System.out.println("  " +
                                                                realCoefficients[i][j] + " + " +
                                                                imaginaryCoefficients[i][j] + "i");
241                                             }
242                                     }
243                             }
244                     }
```

```
245                     catch (Exception ex)
246                     {
247                             ex.printStackTrace();
248                     }
249
250                     out.close();
251                     in.close();
252             }
253             catch (Exception e)
254             {
255                     e.printStackTrace();
256             }
257     }
258
259     //read in specified number of bytes and turn them into some number of doubles
260     public static double[] readRecord(int recSize, int numDoubles, InputStream in, ByteBuffer bIn,
        DoubleBuffer dIn)
261     {
262             double[] rec = new double[numDoubles];
263
264             try
265             {
266                     bIn.position(0);
267                     for (int byteNum = 0; byteNum < recSize; byteNum ++)
268                             bIn.put((byte)in.read());
269
270                     dIn.position(0);
271                     for (int doubleNum = 0; doubleNum < numDoubles; doubleNum ++)
272                             rec[doubleNum] = dIn.get();
273             }
274             catch (Exception ex)
275             {
276                     ex.printStackTrace();
277             }
278
279             return rec;
280     }
281
282     //read in specified number of bytes and turn them into some number of floats
283     public static float[] readRecordF(int recSize, int numFloats, InputStream in, ByteBuffer bIn,
        FloatBuffer fIn)
284     {
285             float[] rec = new float[numFloats];
286
287             try
288             {
289                     bIn.position(0);
290                     for (int byteNum = 0; byteNum < recSize; byteNum ++)
291                             bIn.put((byte)in.read());
292
293                     fIn.position(0);
294                     for (int floatNum = 0; floatNum < numFloats; floatNum ++)
295                             rec[floatNum] = fIn.get();
296             }
297             catch (Exception ex)
298             {
299                     ex.printStackTrace();
300             }
301
302             return rec;
303     }
304
305     //convert an array of doubles into records of bytes
306     public static void writeRecord(int recSize, double[] rec, OutputStream out, ByteBuffer bOut,
        DoubleBuffer dOut)
307     {
308             int numDoubles = rec.length;
309
310             try
```

```
311                 {
312                         bOut.position(0);
313                         for (int byteNum = 0; byteNum < recSize; byteNum ++)
314                                 bOut.put((byte)0);
315
316                         dOut.position(0);
317                         for (int doubleNum = 0; doubleNum < numDoubles; doubleNum ++)
318                                 dOut.put(rec[doubleNum]);
319
320                         out.write(bOut.array());
321                 }
322                 catch (Exception ex)
323                 {
324                         ex.printStackTrace();
325                 }
326         }
327
328         //convert an array of floats into records of bytes
329         public static void writeRecordF(int recSize, float[] rec, OutputStream out, ByteBuffer bOut,
          FloatBuffer fOut)
330         {
331                 int numFloats = rec.length;
332
333                 try
334                 {
335                         bOut.position(0);
336                         for (int byteNum = 0; byteNum < recSize; byteNum ++)
337                                 bOut.put((byte)0);
338
339                         fOut.position(0);
340                         for (int floatNum = 0; floatNum < numFloats; floatNum ++)
341                                 fOut.put(rec[floatNum]);
342
343                         out.write(bOut.array());
344                 }
345                 catch (Exception ex)
346                 {
347                         ex.printStackTrace();
348                 }
349         }
350 }
351
```