```java
import java.util.Scanner;
class Problem3
{
  static double[] wt;
  static double[] mu;
  static int nang;
  public static void main(String[] args)
  {
    try
    {
//*************************************************************************
//                                                                        *
//       1D SN calculation of a slab:                                     *
//                                                                        *
//       |      |                g                   |                    *
//       |      |        -----------------           |                    *
//       |      |        Grp 1       Grp 2           |                    *
//       |      | Total    0.1         0.2           |                    *
//       |      | G1->g    0.05        0.04          |                    *
//       |S1 .4 | G2->g    0           0.1           |                    *
//       |S2 .6 |                                    |                    *
//       |      |                                    |                    *
//       |      |                                    |                    *
//       |      |                                    |                    *
//       |      |                                    |                    *
//       | 5 cm |              45 cm                 |                    *
//       |<---->|<---------------------------------->|                    *
//       |      |                                    |                    *
//                                                                        *
//*************************************************************************
//                                                                        *
//     Variable definitions:                                              *
//       totxs(ig)   Total cross section in group ig, 1/cm                *
//       scat(ig,jg) Scattering cross section from group ig to jg, 1/cm *
//       totscat(ig) Total scattering in group ig (i.e., sum of scat      *
//                      for all other groups jg                           *
//       sour(ig)    Source in group ig, #/cm3/sec                        *
//       bin(ib)     Bin values                                           *
//                   ib = 1 Left leakage for group 1                      *
//                      = 2 Left leakage for group 2                      *
//                      = 3 Right leakage for group 1                     *
//                      = 4 Right leakage for group 2                     *
//                      = 5 Flux for group 1                              *
//                      = 6 Flux for group 2                              *
//       ig          Current energy group of the particle                 *
//       mu          Current direction cosine of the particle, (-1,1)     *
//       x           Current position of particle                         *
//       dd          Distance to next collision                           *
//       dx          x dimension distance to next collision = dd*mu       *
//       mfp         Mean free paths to next collision                    *
//                                                                        *
```

```
//*********************************************************************
//                                                                    *
//       Set the source and cross sections                            *
//                                                                    *
//*********************************************************************
      double[] totxs={0.1,0.2};
      double alpha=0.8;
      double[] left=new double[2];
      double[] right=new double[2];
      double[][] scat=new double[2][2];
      scat[0][0]=.05;
      scat[0][1]=0.04;
      scat[1][0]=0.;
      scat[1][1]=0.1;
      double[] sour={.4,.6};
      sour[0]/=5.;
      sour[1]/=5.;
//*********************************************************************
//                                                                    *
//       Find total scattering cross section for each group           *
//                                                                    *
//*********************************************************************
      int ng=2;
      //System.out.println(" No. of spatial divisions in (0-5)?");
      //Scanner sc=new Scanner(System.in);
      //int ns=sc.nextInt();
      int ns=1000000;
      int nx=ns*10;
      double dx=50./nx;
      //System.out.println(" No. of angles?");
      //nang=sc.nextInt();
      nang=12;
      mu=new double[nang];
      wt=new double[nang];
      setQuadrature();
      double[][] scalar=new double[nx][ng];
      for(int ix=0;ix<nx;ix++)
      {
        for(int ig=0;ig<ng;ig++)
        {
          scalar[ix][ig]=0.;
        }
      }
      double[] sext=new double[nx];
      double[] sourin=new double[nx];
//********************************************************************
//                                                                    *
// Outer iterations: Loop over each group                             *
//                                                                    *
//********************************************************************
      for(int ig=0;ig<ng;ig++)
      {
//************************************************************************
//                                                                       *
//     Find the external and scattering source from other groups        *
//                                                                       *
//************************************************************************
        for(int ix=0;ix<nx;ix++)
        {
          sext[ix]=0.;
          if(ix < ns)sext[ix]+=sour[ig];
          for(int igp=0;igp<ng;igp++)
          {
            if(ig != igp)sext[ix]+=scalar[ix][igp]*scat[igp][ig];
          }
        }
//************************************************************************
//                                                                       *
//  Inner iterations                                                     *
//                                                                       *
//************************************************************************
        int inner=0;
        double eps=0.00001;
        double conv=10000.;
        double[] scalarOld=new double[nx];
        while(Math.abs(conv)>eps)
        {
          inner++;
//************************************************************************
//                                                                       *
//    Add within-group scattering source                                 *
//                                                                       *
//************************************************************************
          for(int ix=0;ix<nx;ix++)
          {
            sourin[ix]=sext[ix]+scalar[ix][ig]*scat[ig][ig];
            scalarOld[ix]=scalar[ix][ig];
            scalar[ix][ig]=0.;
          }
          left[ig]=0.;
          right[ig]=0.;
//************************************************************************
//                                                                       *
//    Loop over directions                                               *
//                                                                       *
//************************************************************************
          for(int ia=0;ia<nang;ia++)
          {
//************************************************************************
//                                                                       *
//      Loop over positions                                              *
//                                                                       *
```

```
//************************************************************************
            double phi0=0.;
            double muabs=Math.abs(mu[ia]);
            for(int ix0=0;ix0<nx;ix0++)
            {
              int ix=ix0;
              if(mu[ia]<0.)ix=nx-1-ix0;
//************************************************************************
//                                                                       *
//        AUXILIARY: Find angular flux for cell and outgoing             *
//                                                                       *
//************************************************************************
//                                                                       *
//          phi0 = Incoming angular flux                                 *
//          phi1 = Outgoing angular flux                                 *
//       fluxave = Average angular flux in cell                          *
//    sourin[ix] =  Source in the cell                                   *
//            mu = Absolute value of cosine of direction                 *
//            dx = Width of the cell                                     *
//     totxs[ig] = Total cross section                                   *
//                                                                       *
//************************************************************************
              double phi1=(sourin[ix] + (muabs / dx - (1. - alpha) * totxs[ig])
               * phi0) / (muabs / dx + alpha * totxs[ig]);
              double fluxave=(1. - alpha) * phi0 + alpha * phi1;
              phi0=phi1;
//************************************************************************
//                                                                       *
//        Add to scalar flux                                             *
//                                                                       *
//************************************************************************
              scalar[ix][ig]+=wt[ia]*fluxave;
            }
//************************************************************************
//                                                                       *
//      Add to outgoing leakage                                          *
//                                                                       *
//************************************************************************
            if(mu[ia]<0.)left[ig]-=wt[ia]*phi0*mu[ia];
            if(mu[ia]>0.)right[ig]+=wt[ia]*phi0*mu[ia];
          }
//************************************************************************
//                                                                       *
//    Check inner convergence                                            *
//                                                                       *
//************************************************************************
          conv=0.;
          for(int ix=0;ix<nx;ix++)
          {
            double etry=(scalar[ix][ig]-scalarOld[ix])/scalar[ix][ig];
            if(Math.abs(etry)>conv)conv=Math.abs(etry);
          }
        }
      }
//************************************************************************
//                                                                       *
//    Print results                                                      *
//                                                                       *
//************************************************************************
      if(true)
      {
        for(int ig=0;ig<ng;ig++)
        {
          System.out.println("FOR GROUP "+(ig+1));
          for(int ix=0;ix<nx;ix++)
          {
            //System.out.println("  Flux pt "+(ix+1)+" = "+scalar[ix][ig]);
          }
          System.out.println(" average of 45-50");
          double avephi=0;
          for(int ix=nx-ns;ix<nx;ix++)
          {
            avephi+=scalar[ix][ig]/ns;
          }
          System.out.println("  Ave "+avephi);
        }
      }
      for(int ig=0;ig<ng;ig++)
      {
        //System.out.println("  Left grp "+(ig+1)+" is "+left[ig]);
      }
      for(int ig=0;ig<ng;ig++)
      {
        //System.out.println(" Right grp "+(ig+1)+" is "+right[ig]);
      }
    }
    catch(Exception e)
    {
      e.printStackTrace(System.out);
    }
  }

  static void setQuadrature() throws Exception
  {
    if(nang==2)
    {
      wt[0]=1.;
      mu[0]=.5773502691;
    }
    else if(nang==4)
    {
      wt[0]=.6521451549;
      wt[1]=.3478548451;
      mu[0]=.3399810435;
      mu[1]=.8611363115;
    }
    else if(nang==8)
    {
      wt[0]=.3626837834;
      wt[1]=.3137066459;
      wt[2]=.2223810344;
      wt[3]=.1012285363;
      mu[0]=.1834346424;
      mu[1]=.5255324099;
      mu[2]=.7966664774;
      mu[3]=.9602898564;
    }
    else if (nang==12)
    {
      wt[0] = 0.04717534;
      wt[1] = 0.10693933;
      wt[2] = 0.16007833;
      wt[3] = 0.20316743;
      wt[4] = 0.23349254;
      wt[5] = 0.24914705;

      mu[0] = 0.98156063;
      mu[1] = 0.90411726;
      mu[2] = 0.76990267;
      mu[3] = 0.58731795;
      mu[4] = 0.3678315;
      mu[5] = 0.12523341;
    }
    else
    {
      throw new Exception(" Quadrature order must be 2,4 or 8");
    }
    double tot=0.;
    for(int ia=0;ia<nang/2;ia++)
    {
      mu[ia+nang/2]=-mu[ia];
      wt[ia]/=2.;
      wt[ia+nang/2]=wt[ia];
      tot+=wt[ia]+wt[ia+nang/2];
    }
    if(Math.abs(tot-1.)>.00001)throw new Exception("Wts add to "+tot);
  }
}
```