MODULE 10 - SHEET 1

```
public class DiningPhilosophers
 { private static final int SIZE = 5;

   public static void main(String[] args)
    { Fork[] forks = new Fork[SIZE];
      for (int f = 0; f<SIZE; f++)
         forks[f] = new Fork(f);
      Thread[] phil = new Thread[SIZE];
      for (int p = 0; p<SIZE; p++)
         phil[p] = new Philosopher(p, forks[p<SIZE-1 ? p+1 : SIZE-1],
                                      forks[p<SIZE-1 ? p : 0]);
      System.out.println(" Philosopher 0   Philosopher 1  " +
                         " Philosopher 2   Philosopher 3   Philosopher 4\n");
      phil[2].start(); phil[1].start(); phil[4].start(); phil[0].start(); phil[3].s
    }
 }

class TAB
 { public static String tab(int n)
    { String s = "";
      for (int i=0; i<n; i++)
         s += "                ";
      return s;
    }
 }

class Fork
 { private int number;
   private boolean inuse = false;

   public Fork(int n)
    { this.number = n;
    }

   public synchronized void get(int n) throws InterruptedException
    { while (this.inuse)
       { System.out.println(TAB.tab(n) + "awaiting fork " + this.number);
         this.wait();
```

```
       }
      System.out.println(TAB.tab(n) + "acquires fork " + this.number);
      this.inuse = true;
      this.notify();
    }

   public synchronized void put(int n) throws InterruptedException
    { while (!this.inuse)
        this.wait();
      System.out.println(TAB.tab(n) + "releases fork " + this.number);
      this.inuse = false;
      this.notify();
    }
 }

class Philosopher extends Thread
 { private int number;
   private Fork first;
   private Fork second;

   public Philosopher(int n, Fork ff, Fork sf)
    { this.number = n;
      this.first = ff;
      this.second =  sf;
    }

   public void run()
    { try
       { for (int i=0; i<3; i++)
          { System.out.println(TAB.tab(this.number) + "wants some food");
            this.first.get(this.number);
            this.second.get(this.number);
            System.out.println(TAB.tab(this.number) + "starts his meal");
            for (int j=0; j<(int)(Math.random()*3); j++)
             { System.out.println(TAB.tab(this.number) + "is still eating");
               this.sleep(1000);
             }
            System.out.println(TAB.tab(this.number) + "has become full");
            this.second.put(this.number);
            this.first.put(this.number);
            System.out.println(TAB.tab(this.number) + "resumes thought");
            this.sleep(3000);
          }
```

```
       }
      catch (InterruptedException e) {}
    }
 }
```

MODULE 10 - SHEET 2

```
public class Hash
 { private static double[] data = {637.42d, 6300.95d, 7.81d, 6300.95d,
                                   712.72d, 4325.22d, 2.79d, 3125.77d,
                                   813.02d, 3125.77d, 6.42d, 1234.56d};
   private static double[] table = new double[630];
   private static int duplicates;

   static
    { for (int i=0; i<table.length; i++)
        table[i] = -1.0d;
      duplicates = 0;
    }

   public static void main(String[] args)
    { for (int i=0; i<data.length; i++)
       { double x = data[i];
         int n = (int)x % 630;
         while (true)
  { if (table[n]<0.0d)
             { table[n] = x;
               break;
             }
            else
               if (table[n] == x)
                { duplicates++;
                  break;
                }
               else
                  n = (n+1) % 630;
  }
      }
      System.out.println("There are " + duplicates + " duplicates");
    }
 }
```

MODULE 10 - SHEET 3

```
public class SolitaireB
 { private static int[] triple = {007000, 000700, 000340, 000034, 000016, 000007,
                                  001104, 012200, 002210, 064000, 024400, 004420,
                                  004204, 002102, 022100, 001041, 011040, 051000};

   private static int[] ta =     {006000, 000600, 000300, 000030, 000014, 000006,
                                  001100, 012000, 002200, 060000, 024000, 004400,
                                  000204, 000102, 002100, 000041, 001040, 011000};

   private static int[] tb =     {003000, 000300, 000140, 000014, 000006, 000003,
                                  000104, 002200, 000210, 024000, 004400, 000420,
                                  004200, 002100, 022000, 001040, 011000, 050000};

   private static final int first = 037777, last = 040000;

   private static int[] scoreArr = new int[0100000];

   public static void main(String[] args)
    { long start = System.currentTimeMillis();
      for (int i=0; i<0100000; i++)
         scoreArr[i] = -1;
      scoreArr[last] = 1;
      System.out.println("There are " + tryit(first) + " solutions");
      long timeSpent = System.currentTimeMillis() - start;
      System.out.println("Done in " + timeSpent + " milliseconds");
    }

   private static int tryit(int state)
    { int score = scoreArr[state];
      if (score < 0)
```

```
      { score = 0;
        for (int i=0; i<18; i++)
         { int x = state & triple[i];
           if (x == ta[i] || x == tb[i])
      score += tryit(state ^ triple[i]);
         }
        scoreArr[state] = score;
      }
     return(score);
   }
 }

// The above program counts the number of ways of solving the Triangular
// Solitaire Problem fairly efficiently.  The program recognises that it
// is a lattice which is being searched for solutions and not a tree.

// There are only 2^15 possible positions and an array of this size is
// set up which, eventually, shows the number of routes to the solution
// from each position.  Initally, every node is set to -1 except the end
// node which corresponds to the finish state.

// The program duly finds the 6816 solutions in about 85 milliseconds.
```