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.printf("  Philosopher 0   Philosopher 1 " +
                        "  Philosopher 2   Philosopher 3   Philosopher 4%n%n");
      phil[2].start(); phil[1].start(); phil[4].start(); phil[0].start(); phil[3].start();
    }
 }

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.printf("%s awaiting fork %d%n", TAB.tab(n), this.number);
         this.wait();
       }
      System.out.printf("%s acquires fork %d%n", TAB.tab(n), this.number);
      this.inuse = true;
      this.notify();
    }

   public synchronized void put(int n) throws InterruptedException
    { while (!this.inuse)
        this.wait();
      System.out.printf("%s releases fork %d%n", TAB.tab(n), 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.printf("%s wants some food%n", TAB.tab(this.number));
            this.first.get(this.number);
            this.second.get(this.number);
            System.out.printf("%s starts his meal%n", TAB.tab(this.number));
            for (int j=0; j<(int)(Math.random()*3); j++)
             { System.out.printf("%s is still eating%n", TAB.tab(this.number));
               this.sleep(1000);
             }
            System.out.printlf("%s has become full%n", TAB.tab(this.number));
            this.second.put(this.number);
            this.first.put(this.number);
            System.out.printf("%s resumes thought%n", TAB.tab(this.number));
            this.sleep(3000);
          }
       }
      catch (InterruptedException e) {}
    }
 }
```

```
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;     // The RHS is a  hash function
         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.printf("There are %s duplicates%n", duplicates);
    }
 }

// The above program implements a  hash table  (the array table) which
// happens to have 630 elements.  Using a for-loop in the  static initializer
// each element is initalized to -1.0d to indicate 'empty'.
//
// Test data is held in the array  data  and each new value is assigned to
// a variable x which is converted via a  hash function  into an int which
// is guaranteed to be in the range 0 to 629.  The hash function of the first
// value 637.42 is 7 for example.
//
// In most cases the value of x is stored in the element of the hash table
// indexed by the hash function value.  Thus 637.42 is stored in table[7].
//
// If the element is not empty then a check is made to see whether it holds
// the current value.  If so, a duplicate has been detected.
//
```

```
// The element may be occupied by a non-duplicate.  This first happens in the
// case of 7.81 which has the same hash function as 637.42 and in such a case
// the next element is tried instead.  The assignment of (n+1)%630 ensures
// that element 0 is treated as following element 629.
//
// As the hash table is becomes fuller, it may be necessary to advance two or
// more cells to find one that is empty.
```