

MODULE 8 - SHEET 1

```
public class ThreadIntro
{ public static void main(String[] args)
  { ThreadJack jack = new ThreadJack();
    ThreadJill jill = new ThreadJill();
    jack.start();
    jill.start();
  }
}

class ThreadJack extends Thread
{ public void run()
  { for (int i=0; i<10; i++)
    { System.out.println("Jack " + i);
      try
    { this.sleep(1000L);
      }
      catch(InterruptedException e)
    {}
  }
}

class ThreadJill extends Thread
{ public void run()
  { for (int i=0; i<10; i++)
    { System.out.println("Jill " + i);
      try
    { this.sleep(500L);
      }
      catch(InterruptedException e)
    {}
  }
}

// This yields:
//
// Jack 0
```

```
// Jill 0
// Jill 1
// Jack 1
// Jill 2
// Jill 3
// Jack 2
// Jill 4
// Jill 5
// Jack 3
// Jill 6
// Jill 7
// Jack 4
// Jill 8
// Jill 9
// Jack 5
// Jack 6
// Jack 7
// Jack 8
// Jack 9
```

MODULE 8 - SHEET 2

```
public class BoxConcluded
{
    public static void main(String[] args)
    {
        Square jack = new Square(6);
        System.out.println("Details of jack...\n" + jack.toString());
        Square jill = new Square(5);
        System.out.println("Details of jill...\n" + jill);
        System.out.println("Number of Squares: " + Square.total + "\n");
        jack = jill;
        jill = null;
        System.gc();
        System.out.println("Number of Squares: " + Square.total + "\n");
        System.out.println("Details of jack...\n" + jack);
    }
}

class Square
{
    public static int total = 0;           // record the number of Squares...

    private int side;

    public Square(int e)
    {
        this.side = e;
        this.total++;                     // ...by incrementing total.
    }

    public int area()
    {
        return this.side*this.side;
    }

    public String toString()
    {
        return "Square: Side = " + this.side + "\n" +
               "          Area = " + this.area() + "\n";
    }
}
```

```

    public void finalize()
    { this.total--;
      }
}

// This yields:
//
// Details of jack...
// Square: Side = 6
//         Area = 36
//
// Details of jill...
// Square: Side = 5
//         Area = 25
//
// Number of Squares: 2
//
// Number of Squares: 1
//
// Details of jack...
// Square: Side = 5
//         Area = 25

```

MODULE 8 - SHEET 3

```
public class ReverseList
{ public static void main(String[] args)
  { Link start = new Link(16);
    start.put(8);
    start.put(4);
    start.put(64);
    start.put(32);
    System.out.println("List elements:  " + start);
    System.out.println("Element sum:    " + start.sum());
    Link rev = start.reverse();
    System.out.println("List elements:  " + rev);
    System.out.println("Element sum:    " + rev.sum());
  }
}

class Link
{ private int val;
  private Link next;

  public Link(int n)
  { this.val = n;
    this.next = null;
  }

  public void put(int k)
  { if (this.next == null)
    { this.next = new Link(k);
    }
    else
    { this.next.put(k);
    }
  }
}
```

```

public Link reverse()
{ if (this.next == null)
    return this;
  else
    { Link temp = this.next.reverse();
      temp.put(this.val);
      return temp;
    }
}

public String toString()
{ return this.val + (this.next == null ? "" : " " + this.next.toString());
}

public int sum()
{ return this.val + (this.next == null ? 0 : this.next.sum());
}
}

```

MODULE 8 - SHEET 4

```
public class TreeSort
{
    public static void main(String[] args)
    {
        Node tree = new Node(16);
        tree.put(8);
        tree.put(4);
        tree.put(64);
        tree.put(32);
        System.out.println("Tree elements: " + tree);
        System.out.println("Element sum: " + tree.sum());
    }
}

class Node
{
    private Node left;
    private int val;
    private Node right;

    public Node(int n)
    {
        this.left = null;
        this.val = n;
        this.right = null;
    }

    public void put(int k)
    {
        if (k < this.val)
            if (this.left == null)
```

```

        this.left = new Node(k);
    else
        this.left.put(k);
    else
        if (this.right == null)
            this.right = new Node(k);
        else
            this.right.put(k);
    }

    public String toString()
    { String s = ((this.left != null) ? this.left.toString() : "") +
        this.val + " " +
        ((this.right != null) ? this.right.toString() : "");
        return s;
    }

    public int sum()
    { int s = ((this.left != null) ? this.left.sum() : 0) +
        this.val +
        ((this.right != null) ? this.right.sum() : 0);
        return s;
    }
}

```

MODULE 8 - SHEET 5

```
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.IOException;

public class WordCount
{ private static final DataInputStream DIS = new DataInputStream(System.in);
  private static int count=0;

  public static void main(String[] args)
  { try
    { while (true)
      { String s = readWord();
      count++;
      }
    }
  catch(EOFException e)
  { System.out.println("Total number of words: " + count);
  }
  catch(IOException e)
  { System.out.println("IOException Encountered");
  }
}
```

```

private static String readWord() throws EOFException, IOException
{
    char ch;
    String s="";
    do
    {
        ch = (char)DIS.readByte();
    } while (ch==' ' || ch=='\t' || ch=='\n');
    do
    {
        s += ch;
        ch = (char)DIS.readByte();
    } while (ch!=' ' && ch!='\t' && ch!='\n');
    return s;
}
}

// Example run:
//
// $ javac WordCount.java
// The quick
// brown fox
// jumps over
// the lazy dog.
// ^d
//
// Total number of words: 9

```

MODULE 8 - SHEET 6

```
public class CroquetA
{ private static int count=0;
  private static int[] plan = new int[28];
  private static boolean[] alreadyPlayed = new boolean[193];
  private static final int p1 = 1, p2 = 2, p3 = 4, p4 = 8,
                          p5 =16, p6 =32, p7 =64, p8 =128;
  private static final int maxgame = 28;

  public static void main(String[] args)
  { for (int i=0; i<=192; i++)
      alreadyPlayed[i] = false;
    tryit(0);
    System.out.println("There are " + count + " solutions");
  }

  private static void tryit(int game)
  { if (game == maxgame)
      { count++;
```

```

        printOut();
        System.exit(0);    // Abort after printing first solution
        return;
    }
    int imposs = 0;
    for (int i = game & 0x1C; i<game; i++)
        imposs |= plan[i];
    int poss = ~imposs & 0xFF;
    int[] player = new int[8];
    int k = -1;
    while (poss!=0)
    { player[++k] = poss & -poss;
      poss &= ~player[k];
    }
    for (int i=0; i<k; i++)
        for (int j=i+1; j<=k; j++)
        { int pi = player[i];
          int pj = player[j];
          int pair = pi | pj;
          if (!(alreadyPlayed[pair]))
          { plan[game] = pair;
            alreadyPlayed[pair] = true;
            tryit(game+1);
            alreadyPlayed[pair] = false;
          }
        }
    }

private static void printOut()
{ for (int lawn=0; lawn<4; lawn++)
    { for (int game=lawn; game<maxgame; game += 4)
        System.out.print(match(plan[game]) + " ");
      System.out.println();
    }
}

private static String match(int pair)
{ String s = "";
  for (int p = pair & -pair; pair != 0; p = pair & -pair)
  { switch(p)
    { case p1:  s += 1;  break;
      case p2:  s += 2;  break;
      case p3:  s += 3;  break;
    }
  }
}

```

```

        case p4: s += 4; break;
        case p5: s += 5; break;
        case p6: s += 6; break;
        case p7: s += 7; break;
        case p8: s += 8;
    }
    pair &= ~p;
}
return s;
}
}

```

MODULE 8 - NOT A HANDOUT

```

public class SolitaireA
{ private static int count=0;

    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;

public static void main(String[] args)
{ long start = System.currentTimeMillis();
  tryit(first);
  long timeSpent = System.currentTimeMillis() - start;
  System.out.println("There are " + count + " solutions");
  System.out.println("Done in " + timeSpent + " milliseconds");
}

private static void tryit(int state)
{ if (state == last)
  { count ++;
    return;
  }
  for (int i=0; i<18; i++)
  { int x = state & triple[i];
    if (x == ta[i] || x == tb[i])
      tryit(state ^ triple[i]);
  }
}
}

// The above program inefficiently counts the number of ways of solving
// the Triangular Solitaire Problem which is played on a board which has
// 15 holes:
//
//
//          1
//         2  3
//        4  5  6
//       7  8  9 10
//      11 12 13 14 15
//
// The program correctly discovers that there are 6816 solutions but, when
// timed, took 25009 milliseconds.

```