Inf1 OOP Main Exam: Sample Solutions

Given the limited nature of the tasks in these questions, it is difficult for students to demonstrate excellence over and above giving correct answers. Therefore, in general a set of answers that produces the correct results will receive full marks.

Answers which fail to produce correct results will still be given partial credit, according to the following guidelines:

- If there appears to be a good understanding of the logic of the task, but there are minor syntactic errors, then 25% will be deducted from the marks available.
- If there is a significant error in the logic of the task, then 50% will be deducted from the marks available.

Obviously, these penalties will be adjusted according to the severity of the problem.

Question 1 Solutions

In some of the questions, and the sample solutions, the variable `N` is used to indicate the length of arrays. Although not standard practice in Java, this follows the convention used in the course textbook (and some of the lectures).

`OneA.java`

```
public class OneA {

    public static int prodOfPairs(int[] nums) {
        int sum = 0;
        if (nums.length % 2 != 0)
            return -1;
        for (int i = 0; i < nums.length - 1; i = i + 2) {
            int prod = nums[i] * nums[i + 1];
            sum += prod;
        }
        return sum;
    }



}
```

OneB.java

```
public class OneB {

    public static double meanColSums(int[][] matrix) {
        double sum = 0.0;
        int rowLength = matrix[0].length;
        for (int i = 0; i < rowLength; i++) {
            for (int j = 0; j < matrix.length; j++) {
                sum += matrix[j][i];
            }
        }
        sum /= rowLength;
        return sum;
    }
}
```

OneC.java

```java
import java.util.ArrayList;

public class OneC {

    public static int hammingDist(String left, String right) {
        int diffs = 0;

        for (int i = 0; i < left.length(); i++) {
            if (left.charAt(i) != right.charAt(i))
                diffs++;
        }
        return diffs;
    }

    public static String findFarthest(String s, String[] targets) {
        String farthest = s;
        int dist = 0;
        for (String t : targets) {
            int d = hammingDist(s, t);
            if (d > dist) {
                dist = d;
                farthest = t;
            }
        }
        return farthest;
    }

    public static ArrayList<String> findNearestK(String s, String[] targets,
            int k) {
        ArrayList<String> cluster = new ArrayList<String>();
        for (String t : targets) {
            int d = hammingDist(s, t);

            if (d <= k) {
                cluster.add(t);
            }
        }
        return cluster;
    }

    public static int stringDist(String left, String right) {
        int penalty = 0;
        int llen = left.length();
        int rlen = right.length();
        if (rlen > llen) {
            right = right.substring(0, llen);
            penalty = rlen - llen;
        }
        if (llen > rlen) {
            left = left.substring(0, rlen);
            penalty = llen - rlen;
        }

        return hammingDist(left, right) + penalty;
    }

}
```

QuestionOneTester.java

The main rationale for this question is to encourage students to budget some time for testing. They have seen many examples of testing code which runs some methods inside `main()`. If they have two reasonable-looking calls for each of their three methods, they will receive full marks.

Question 1 Supplied Files

The following skeleton and data files are provided to the students.

OneA.java

```
public class OneA {

    public static int prodOfPairs(int[] nums) {
        // ADD CODE HERE
    }



}
```

OneB.java

```
public class OneB {

    public static double meanColSums(int[][] matrix) {
        // ADD CODE HERE
    }
}
```

OneC.java

```java
import java.util.ArrayList;
import java.util.Arrays;

public class OneC {

    public static int hammingDist(String left, String right) {
        // ADD CODE HERE
    }

    public static String findFarthest(String s, String[] targets) {
        // ADD CODE HERE
    }

    public static ArrayList<String> findNearestK(String s, String[] targets,
            int k) {
        // ADD CODE HERE
    }

    public static int stringDist(String left, String right) {
        // ADD CODE HERE
    }

}
```

Question 2 Solutions

Expr.java

```
public abstract class Expr {


    public abstract boolean isTerm();

    public abstract boolean isNorm();

    public Expr getLeft() {
        return null;
    }

    public Expr getRight() {
        return null;
    }

    public Op getOp() {
        return null;
    }


    public Expr normalize() {
        return this;
    }



}
```

Var.java

```
public class Var extends Expr {

    String symbol;

    public Var(String symbol) {
        this.symbol = symbol;
//        setTerm(true);
//        setNorm(true);
    }

    public boolean isTerm(){
        return true;
    }

    public boolean isNorm(){
        return true;
    }


    public String toString() {
        return symbol;
    }



}
```

BinaryExpr.java

```
public class BinaryExpr extends Expr {

private Expr left;
private Expr right;
private Op op;

public BinaryExpr(Expr left, Op op, Expr right) {
    this.left = left;
    this.right = right;
    this.op = op;
}

public Expr getLeft() {
    return left;
}

public Expr getRight() {
    return right;
}

public Op getOp() {
    return op;
}

public boolean isTerm() {
    return left.isTerm() && op == Op.PRODUCT && right.isTerm();
}

public boolean isNorm() {
    boolean b1 = this.isTerm();
    boolean b2 = left.isNorm() && op == Op.SUM && right.isNorm();
    return b1 || b2;
}

//@Override
public Expr normalize() {

    if (op == Op.PRODUCT && left != null) {
        if (left.getOp() == Op.SUM) {
            BinaryExpr l = new BinaryExpr(left.getLeft(), Op.PRODUCT, right);
            BinaryExpr r = new BinaryExpr(left.getRight(), Op.PRODUCT,
                    right);
            return new BinaryExpr(l.normalize(), Op.SUM, r.normalize());
        }
        if (right.getOp() == Op.SUM) {
            BinaryExpr l = new BinaryExpr(left, Op.PRODUCT, right.getLeft());
            BinaryExpr r = new BinaryExpr(left, Op.PRODUCT,
                    right.getRight());
            return new BinaryExpr(l.normalize(), Op.SUM, r.normalize());
        }
    }
    return this;
}

public String toString() {
    String s = String.format("(%s %s %s)", left, op, right);
    return s;
    }

}
```

Question 2 Supplied files

`Op.java`

```
public enum Op {

    SUM {
        public String toString() {
            return "+";
        }
    },

    PRODUCT {
        public String toString() {
            return "*";
        }
    };

}
```