Department of Computing and Software, McMaster University Instructor: Dr. Reza Samavi
2XB3

Lab 02 Solutions for Practice problems

Topic **Eclipse and Java Setup**

Task 1: Write a Java program that tests whether an integer corresponds to a leap year in the Gregorian calendar.

```
/*****************************************************************
 *  Compilation:  javac LeapYear.java
 *  Execution:    java LeapYear n
 *  Prints true if n corresponds to a leap year, and false
otherwise.
 *  Assumes n >= 1582, corresponding to a year in the Gregorian
calendar.
 *  % java LeapYear 2004
 *  true
 *  % java LeapYear 1900
 *  false
 *  % java LeapYear 2000
 *  true
 *****************************************************************/
public class LeapYear {
    public static void main(String[] args) {
        int year = Integer.parseInt(args[0]);
        boolean isLeapYear;
        // divisible by 4
        isLeapYear = (year % 4 == 0);
        // divisible by 4 and not 100
        isLeapYear = isLeapYear && (year % 100 != 0);
        // divisible by 4 and not 100 unless divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);
        System.out.println(isLeapYear);
    }
}
```

Task 2: Write a Java program that reads an integer command-line argument *n* and prints a "random" integer between 0 and *n*−1. **Hint:** You will be reading the argument as a String.

```
public class RandomInt {
    public static void main(String[] args) {
        // a positive integer
        int n = Integer.parseInt(args[0]);

        // a pseudo-random real between 0.0 and 1.0
        double r = Math.random();

        // a pseudo-random integer between 0 and n-1
        int value = (int) (r * n);
        System.out.println(value);
    }
}
```

Task 3: Write a Java program Flip.java that uses Math.random() and an if-else statement to print the results of a coin flip.

```
public class Flip {

    public static void main(String[] args) {

        // Math.random() returns a value between 0.0 and 1.0
        // so it is heads or tails 50% of the time
        if (Math.random() < 0.5) System.out.println("Heads");
        else                     System.out.println("Tails");
    }
}
```

Task 4: Write a Java program TenHellos.java that prints "Hello World" 10 times.

```
public class TenHellos {
    public static void main(String[] args) {

        // print out special cases whose ordinal doesn't end in th
        System.out.println("1st Hello");
        System.out.println("2nd Hello");
        System.out.println("3rd Hello");

        // count from i = 4 to 10
        int i = 4;
        while (i <= 10) {
            System.out.println(i + "th Hello");
            i = i + 1;
        }

    }
}
```

Task 5: Let's put everything together and write a Java program Deck.java that contains the full code for creating and shuffling a deck of cards.

```
/******************************************************************
 *  Execution:    java Deck
 *  Deal 52 cards uniformly at random.
 *  % java Deck
 *  Ace of Clubs
 *  8 of Diamonds
 *  5 of Diamonds
 *  ...
 *  8 of Hearts

*******************************************************************/

public class Deck {
    public static void main(String[] args) {
        String[] SUITS = {
            "Clubs", "Diamonds", "Hearts", "Spades"
        };

        String[] RANKS = {
            "2", "3", "4", "5", "6", "7", "8", "9", "10",
            "Jack", "Queen", "King", "Ace"
        };

        // initialize deck
        int n = SUITS.length * RANKS.length;
        String[] deck = new String[n];
        for (int i = 0; i < RANKS.length; i++) {
            for (int j = 0; j < SUITS.length; j++) {
                deck[SUITS.length*i + j] = RANKS[i] + " of " +
SUITS[j];
            }
        }

        // shuffle
        for (int i = 0; i < n; i++) {
            int r = i + (int) (Math.random() * (n-i));
            String temp = deck[r];
            deck[r] = deck[i];
            deck[i] = temp;
        }

        // print shuffled deck
        for (int i = 0; i < n; i++) {
            System.out.println(deck[i]);
        }
    }

   }
```

Task 6: Write a code fragment Transpose.java to transpose a square two-dimensional array in place without creating a second array.

```
public class Transpose {
    public static void main(String[] args) {

        // create n-by-n matrix
        int n = Integer.parseInt(args[0]);
        int[][] a = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                a[i][j] = n*i + j;
            }
        }

        // print out initial matrix
        System.out.println("Before");
        System.out.println("------");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.printf("%4d", a[i][j]);
            }
            System.out.println();
        }

        // transpose in-place
        for (int i = 0; i < n; i++) {
            for (int j = i+1; j < n; j++) {
                int temp = a[i][j];
                a[i][j] = a[j][i];
                a[j][i] = temp;
            }
        }

        // print out transposed matrix
        System.out.println();
        System.out.println("After");
        System.out.println("------");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.printf("%4d", a[i][j]);
            }
            System.out.println();
        }

    }
}
```

4. Practice Problems

1· What do each of the following print?

a. System.out.println(2 + "bc"); prints: 2bc

b. System.out.println(2 + 3 + "bc"); prints: 5bc

c. System.out.println((2+3) + "bc"); prints: 5bc
d. System.out.println("bc" + (2+3)); prints: bc5
e. System.out.println("bc" + 2 + 3); prints: bc23

Explain each outcome.

2. A physics student gets unexpected results when using the code
double force = G * mass1 * mass2 / r * r;
to compute values according to the formula $F = Gm_1m_2 / r^2$. Explain the problem and correct the code.\

sln: *Solution*: It divides by r, then multiplies by r (instead of dividing by r *r). Use parentheses:
double force = G * mass1 * mass2 / (r * r);

3. Write a program SpringSeason.java that takes two int values m and d from the command line and prints true if day d of month m is between March 20 (m = 3, d =20) and June 20 (m = 6, d = 20), false otherwise.

```
/*******************************************************************
***********
 *  Compilation:  javac SpringSeason.java
 *  Execution:    java day month
 *
 *  Prints true if the given day and month fall between March 20
(inclusive)
 *  and June 20 (inclusive).
 *
 *  % java SpringSeason 3 20
 *  true
 *
 *  % java SpringSeason 6 20
 *  true
 *
 *  % java SpringSeason 4 15
 *  true
 *
 *  % java SpringSeason 9 11
 * false
 *

********************************************************************
**********/

public class SpringSeason {
    public static void main(String[] args) {
        int month = Integer.parseInt(args[0]);
        int day   = Integer.parseInt(args[1]);
        boolean isSpring =  (month == 3 && day >= 20 && day <= 31)
                         || (month == 4 && day >=  1 && day <= 30)
                         || (month == 5 && day >=  1 && day <= 31)
                         || (month == 6 && day >=  1 && day <= 20);
        System.out.println(isSpring);
    }
}
```

Loop and conditions

4. Suppose a gambler makes a series of fair $1 bets, starting with $50, and continue to play until she either goes broke or has $250. What are the chances that she will go home with $250, and how many bets might she expect to make before winning or losing? Write a Gambler.java program that is a simulation that can help answer these questions. It takes three command-line arguments, the initial stake ($50), the goal amount ($250), and the number of times we want to simulate the game.

```
/*******************************************************************
***********
 *  Compilation:  javac Gambler.java
 *  Execution:    java Gambler stake goal N
 *
 *  Simulates a gambler who start with $stake and place fair $1 bets
 *  until she goes broke or reach $goal. Keeps track of the number
of
 *  times she wins and the number of bets she makes. Run the
experiment N
 *  times, averages the results, and prints them out.
 *
 *  % java Gambler 50 250 1000
 *  178 wins of 1000
 *  Percent of games won = 17.8
 *  Avg # bets           = 10010.79
 *
 *  % java Gambler 50 150 1000
 *  337 wins of 1000
 *  Percent of games won = 33.7
 *  Avg # bets           = 4863.95
 *
 *  % java Gambler 50 100 1000
 *  503 wins of 1000
 *  Percent of games won = 50.3
 *  Avg # bets           = 2464.59
 *

********************************************************************
**********/

public class Gambler {

    public static void main(String[] args) {
        int stake  = Integer.parseInt(args[0]);    // gambler's
stating bankroll
        int goal   = Integer.parseInt(args[1]);    // gambler's
desired bankroll
        int trials = Integer.parseInt(args[2]);    // number of
trials to perform

        int bets = 0;        // total number of bets made
        int wins = 0;        // total number of games won

        // repeat trials times
        for (int t = 0; t < trials; t++) {

            // do one gambler's ruin simulation
            int cash = stake;
            while (cash > 0 && cash < goal) {
                bets++;
                if (Math.random() < 0.5) cash++;     // win $1
                else                     cash--;     // lose $1
            }
            if (cash == goal) wins++;                // did gambler
go achieve desired goal?
        }

        // print results
        System.out.println(wins + " wins of " + trials);
        System.out.println("Percent of games won = " + 100.0 * wins
/ trials);
        System.out.println("Avg # bets           = " + 1.0 * bets /
trials);
    }

}
```

Arrays

5. ***Sampling without replacement***. In many situations, we want to draw a random sample from a set such that each member of the set appears at most once in the sample. Write a Java program Sample.java that takes two command-line arguments m and n, and creates a *permutation* of length n whose first m entries comprise a random sample. See the textbook for details.

```
/*******************************************************************
***********
 *  Compilation:  javac Sample.java
 *  Execution:    java Sample m n
 *
 *  This program takes two command-line arguments m and n and
produces
 *  a random sample of m of the integers from 0 to n-1.
 *
 *  % java Sample 6 49
 *  10 20 0 46 40 6
 *
 *  % java Sample 10 1000
 *  656 488 298 534 811 97 813 156 424 109
 *

*********************************************************************
**********/

public class Sample {
    public static void main(String[] args) {
        int m = Integer.parseInt(args[0]);    // choose this many 
elements
        int n = Integer.parseInt(args[1]);    // from 0, 1, ..., n-1

        // create permutation 0, 1, ..., n-1
        int[] perm = new int[n];
        for (int i = 0; i < n; i++)
            perm[i] = i;

        // create random sample in perm[0], perm[1], ..., perm[m-1]
        for (int i = 0; i < m; i++)  {

            // random integer between i and n-1
            int r = i + (int) (Math.random() * (n-i));

            // swap elements at indices i and r
            int t = perm[r];
            perm[r] = perm[i];
            perm[i] = t;
        }

        // print results
        for (int i = 0; i < m; i++)
            System.out.print(perm[i] + " ");
        System.out.println();
    }
}
```

6. ***Random walkers.*** Suppose that n random walkers, starting in the center of an n-by-n grid, move one step at a time, choosing to go left, right, up, or down with equal probability at each step. Write a program RandomWalkers.java to help formulate and test a hypothesis about the number of steps taken before all cells are touched.

```
/********************************************************************
***********
```

```
 *  Compilation:  javac RandomWalkers.java
 *  Execution:    java RandomWalker n
 *
 *  Simulates how long it takes n random walkers starting at the
center
 *  of an n-by-n grid to visit every cell in the grid.
 *

*********************************************************************
**********/

public class RandomWalkers {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int[] x = new int[n];         // x positions
        int[] y = new int[n];         // y positions
        int cellsToVisit = n*n;      // cells left to visit
        int steps = 0;               // number of steps taken
        double r;
        boolean[][] visited = new boolean[n][n];  // has the i-j
been visited?

        // start at center
        for (int i = 0; i < n; i++) {
            x[i] = n/2;
            y[i] = n/2;
        }
        visited[n/2][n/2] = true;
        cellsToVisit--;


        // repeat until all cells have been visited
        while (cellsToVisit > 0) {
            steps++;

            // move random walker i
            for (int i = 0; i < n; i++) {
                r = Math.random();
                if      (r <= 0.25) x[i]++;
                else if (r <= 0.50) x[i]--;
                else if (r <= 0.75) y[i]++;
                else if (r <= 1.00) y[i]--;

                // check if (x[i], y[i]) is inside N-by-N boundary
and has been visited
                if (x[i] < n && y[i] < n && x[i] >= 0 && y[i] >= 0
&& !visited[x[i]][y[i]]) {
                    cellsToVisit--;
                    visited[x[i]][y[i]] = true;
                }
            }
        }

        System.out.println(steps);
    }

}
```