Intro2CS HW 03

יואב שרת
1-9-2024

1. Loan calculation

```java
/**
* Computes the periodical payment necessary to re-pay a given loan.
*/
public class LoanCalc {

    static double epsilon = 0.001;  // The computation tolerance (estimation
error)
    static int iterationCounter;    // Monitors the efficiency of the
calculation

    /**
     * Gets the loan data and computes the periodical payment.
     * Expects to get three command-line arguments: sum of the loan (double),
     * interest rate (double, as a percentage), and number of payments
(int).
     */
    public static void main(String[] args) {
        // Gets the loan data
        double loan = Double.parseDouble(args[0]);
        double rate = Double.parseDouble(args[1]);
        int n = Integer.parseInt(args[2]);
        System.out.println("Loan sum = " + loan + ", interest rate = " + rate
+ "%, periods = " + n);

        // Computes the periodical payment using brute force search
        System.out.print("Periodical payment, using brute force: ");
        System.out.printf("%.2f", bruteForceSolver(loan, rate, n, epsilon));
        System.out.println();
        System.out.println("number of iterations: " + iterationCounter);

        // Computes the periodical payment using bisection search
        System.out.print("Periodical payment, using bi-section search: ");
        System.out.printf("%.2f", bisectionSolver(loan, rate, n, epsilon));
        System.out.println();
        System.out.println("number of iterations: " + iterationCounter);
    }

    /**
    * Uses a sequential search method  ("brute force") to compute an
approximation
    * of the periodical payment that will bring the ending balance of a loan
close to 0.
    * Given: the sum of the loan, the periodical interest rate (as a
percentage),
    * the number of periods (n), and epsilon, a tolerance level.
    */
    // Side effect: modifies the class variable iterationCounter.
    public static double bruteForceSolver(double loan, double rate, int n,
double epsilon) {
        double g = loan / n;
        while (endBalance(loan, rate, n, g) >= epsilon) {
            iterationCounter++;
            g += epsilon;
        }
        return g;
    }

    /**
    * Uses bisection search to compute an approximation of the periodical
payment
    * that will bring the ending balance of a loan close to 0.
    * Given: the sum of theloan, the periodical interest rate (as a
percentage),
    * the number of periods (n), and epsilon, a tolerance level.
    */
    // Side effect: modifies the class variable iterationCounter.
    public static double bisectionSolver(double loan, double rate, int n,
double epsilon) {
        iterationCounter = 0;
        double L = (loan/n) , H = loan;
        double g = (L + H) / 2;
        while ((H - L) > epsilon) {
            if ((endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L))
> 0) {
                L = g;
            }
            else {
                H = g;
            }
            g = (L + H) / 2;
            iterationCounter++;
        }
        return g;
    }

    /**
    * Computes the ending balance of a loan, given the sum of the loan, the
periodical
    * interest rate (as a percentage), the number of periods (n), and the
periodical payment.
    */
    private static double endBalance(double loan, double rate, int n, double
payment) {
        double endingBalance = loan;
        while (n > 0)
        {
            endingBalance = ((endingBalance - payment) * (1 + rate / 100));
            n = n-1;
        }
        return endingBalance;
    }
}
```

2. Lower case

```java
/** String processing exercise 1. */
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

   /**
    * Returns a string which is identical to the original string,
    * except that all the upper-case letters are converted to lower-case
letters.
    * Non-letter characters are left as is.
    */
    public static String lowerCase(String s) {
        String output = "";
        int string_len = s.length();

        for(int i=0; i<string_len; i++){
            char curr = s.charAt(i);
            // is a capital letter
            if((int)curr >= 65 && (int)curr <=90){
                output += (char)(curr + 32);
            }
            else{
                output += curr;
            }
        }
        return output;
    }
}
```

3. Unique characters

```java
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    /**
     * Returns a string which is identical to the original string,
     * except that all the duplicate characters are removed,
     * unless they are space characters.
     */
    public static String uniqueChars(String s) {
        String output = "";
        int input_length = s.length();
        for(int i=0; i<input_length; i++){
            char curr = s.charAt(i);
            if(output.indexOf(curr) == -1 || (int)curr == 32){
                output += curr;
            }
        }
        return output;
    }
}
```

4. Calendar

```java
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar {
    // Starting the calendar on 1/1/1900
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;     // 1.1.1900 was a Monday
    static int nDaysInMonth = 31; // Number of days in January

    /**
     * Prints the calendar of an inputed year;
     * If its a Sunday - prints 'Sunday' as well.
     */
    public static void main(String args[]) {
        // get user input
        int user_year = Integer.parseInt(args[0]);
        boolean verbose = (user_year == year);
        while (year <= user_year) {
            if(verbose){
                System.out.print(dayOfMonth + "/" + month + "/" + year);
                if(dayOfWeek == 1){
                    System.out.print(" Sunday");
                }
                System.out.println();
            }

            advance();
            verbose = (user_year == year);
        }
     }

     // Advances the date (day, month, year) and the day-of-the-week.
     // If the month changes, sets the number of days in this month.
     // Side effects: changes the static variables dayOfMonth, month, year,
dayOfWeek, nDaysInMonth.
     private static void advance() {
        //advance day of week
        dayOfWeek = (dayOfWeek == 7) ? 1 : dayOfWeek + 1;
        //advance day of month
        if(dayOfMonth != nDaysInMonth){
            dayOfMonth++;
        } else {
            // advance month
            dayOfMonth = 1;
            month = (month == 12) ? 1 : month + 1;
            if(month == 1){
                // needs to advance year
                year++;
            }
            // update nDaysInMonth
            nDaysInMonth = nDaysInMonth(month, year);
        }
     }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        if(year % 4 != 0)
        return false;
        if(year % 100 != 0)
            return true;
        if(year % 400 == 0)
            return true;
        return false;
    }

    // Returns the number of days in the given month and year.
    // April, June, September, and November have 30 days each.
    // February has 28 days in a common year, and 29 days in a leap year.
    // All the other months have 31 days.
    private static int nDaysInMonth(int month, int year) {
        boolean is_leap = isLeapYear(year);

        switch (month) {
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                return is_leap ? 29 : 28;
            default:
                break;
        }
        return 31;
    }
}
```