```
/**
* 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) {
        iterationCounter = 0;
        double g = (loan / n );
        while ((endBalance(loan, rate, n, g) >= epsilon) && (g <= loan)) {
                g += epsilon;
                iterationCounter++;
        }

        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;
        double H = (loan / n);
        double g = ((L + H)/ 2);
        while ((L - H ) > epsilon) {
                if (endBalance(loan, rate, n, g) < 0) {
                        L = g;
                        g = ((H + L) /2);
                } else {
                        H =g;
                        g = ((H + L) / 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) {
                // Replace the following statement with your code
                double endBalance = loan;
                for (int i = 0; i < n; i++){
                endBalance = (endBalance - payment) * ((rate / 100) + 1);
                }
        return endBalance;
        }
}
```

```
/** 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 ans = "";
                  for (int i = 0; i < s.length(); i++) {
                        char ch = s.charAt(i);
                        if (ch == ' ') {
                              ans = (ans + ch);
                        } else if (ch > 64 && ch < 91) {
                              ans = ans + (char) (ch + 32);
                        } else {
                              ans = ans +ch;
                        }
                  }

      return ans;
   }
```

```
        /** 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 ans = "";
         String used = "";
         char current = '0';
      for (int i = 0; i < s.length(); i++) {
         current = s.charAt(i);
         if (current == 32) {
                  ans = ans + " ";
         } else if (used.indexOf(current) == -1) {
                  ans = ans + current;
                  used = used + current;
         } else {
                  used = used + current;
                        }
      }
      return ans;
   }
}
```

```
/**
 * gets a given year, and prints the calendar of that year
 */
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 a given year, also specifies if the day is Sunday.
	 */
	public static void main(String args[]) {
		int given_year = Integer.parseInt(args[0]);
		int debugDaysCounter = 0;

		while (year <= given_year) {
			if (year == given_year) {
				if (dayOfWeek == 1) {
					System.out.println(dayOfMonth + "/" + month + "/"
+ year + " Sunday");
				} else {
					System.out.println(dayOfMonth + "/" + month + "/"
+ year);
				}
			}

			advance();
			debugDaysCounter++;
```

```
                    if (false) {
                        break;
                    }
    }
        }

        // 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() {
            nDaysInMonth = nDaysInMonth(month,year);
            if (dayOfWeek < 7) {
                dayOfWeek++;
            } else {
                dayOfWeek = 1;
            }

            if (nDaysInMonth == 28) {
                if (dayOfMonth < 28) {
                    dayOfMonth++;
                } else {
                    dayOfMonth = 1;
                    month++;
                }
            } else if (nDaysInMonth == 29) {
                if (dayOfMonth < 29) {
                    dayOfMonth++;
                } else {
                    dayOfMonth = 1;
                    month++;
```

```
                }
            } else if (nDaysInMonth == 30) {
                if (dayOfMonth < 30) {
                    dayOfMonth++;
                } else {
                    dayOfMonth = 1;
                    month++;
                }
            } else {
                if (dayOfMonth < 31) {
                    dayOfMonth++;
                } else {
                    dayOfMonth = 1;
                    month++;
                }
            }

            if (month == 13) {
                month = 1;
                year++;
            }
        }


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

```
        // 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) {
                int numDays = 0;

                switch (month) {
                        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                                numDays = 31;
                                break;
                        case 4: case 6: case 9: case 11:
                                numDays = 30;
                                break;
                        case 2:
                                if(isLeapYear(year)) {
                                        numDays = 29;
                                } else {
                                        numDays = 28;
                                }
                                break;

                }
                return numDays;
        }
}
```