```
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) > 0) {
                  g = 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) {
      double H = loan;
            double L = loan/n;
            double g = (L + H)/2;
            iterationCounter = 0;

            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;
            for (int i = 0; i < n; i++) {
                  endingBalance = ((endingBalance - payment) * rate/100) +
(endingBalance - payment);
            }
```

```
        return endingBalance;
        }

}
```

```
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 lowerString = "";
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) >= 'A' && s.charAt(i) <= 'Z') {
                lowerString = lowerString + (char)(s.charAt(i) + 32);
            } else {
                lowerString = lowerString + s.charAt(i);
            }
        }
        return lowerString;
    }
}
```

```
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 newString = "";

        for (int i = 0; i < s.length(); i++) {
            String currChar = String.valueOf(s.charAt(i));
            if (s.indexOf(currChar) == i || s.charAt(i) == ' ') {
                newString = newString + s.charAt(i);
            }
        }

        return newString;
    }
}
```

```
public class Calendar {
    public static void main(String args[]) {
            int yearInput = Integer.parseInt(args[0]);

        int dayOfMonth = 1;
          int month = 1;
          int year = 1900;
          int dayOfWeek = 2;
        int daysInMonth = 31;

        while (year < yearInput) {
            dayOfWeek++;
            dayOfMonth++;

            daysInMonth = nDaysInMonth(month, year);

            if (dayOfWeek > 7) {
                dayOfWeek = 1;
            }

            if (dayOfMonth > daysInMonth) {
                if (month >= 12) {
                    month = 1;
                    year++;
                } else {
                    month++;
                }
                dayOfMonth = 1;
            }

        }
```

```
        while (yearInput != -1) {
            switch (dayOfWeek) {
                case 1:
                    System.out.println(dayOfMonth + "/" + month + "/" +
year + " Sunday");
                    break;

                default:
                    System.out.println(dayOfMonth + "/" + month + "/" +
year);
                    break;
                }

            dayOfWeek++;
            dayOfMonth++;

            daysInMonth = nDaysInMonth(month, year);

            if (dayOfWeek > 7) {
                dayOfWeek = 1;
            }

           if (dayOfMonth > daysInMonth) {
                if (month >= 12) {
                    yearInput = -1;
                } else {
                    month++;
                }
                dayOfMonth = 1;
            }
        }

    }
```

```
    private static boolean isLeapYear(int year) {
          boolean isLeap = false;

            if (((year % 400) == 0) || (((year % 4) == 0) && ((year % 100)
!= 0))) {
                  isLeap = true;
            }
            return isLeap;
      }

      // Returns the number of days in the given month and year.
      private static int nDaysInMonth(int month, int year) {
            int days = 0;

            switch (month) {
                  case 1:
                        days = 31;
                        break;
                  case 2:
                        if (isLeapYear(year)) {
                              days = 29;
                        } else {
                              days = 28;
                        }
                        break;
                  case 3:
                        days = 31;
                        break;
                  case 4:
                        days = 30;
                        break;
```

```
                case 5:
                        days = 31;
                        break;
                case 6:
                        days = 30;
                        break;
                case 7:
                        days = 31;
                        break;
                case 8:
                        days = 31;
                        break;
                case 9:
                        days = 30;
                        break;
                case 10:
                        days = 31;
                        break;
                case 11:
                        days = 30;
                        break;
                case 12:
                        days = 31;
                        break;
            }
            return days;
      }
}
```