```
public class Calendar {
    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);
        advance(year);
}
    private static void advance(int stopyear) {
          int year=1900;
          int sundaycounter=1;
        boolean stop= true;
          while (stop)
          {
               for(int i=1;i<13;i++)
              {
               for(int j=1;j<=nDaysInMonth(i,year);j++)
               {
                sundaycounter++;
                   if (i == 12 && j == nDaysInMonth(i,year))
                    {
                         year++;
                         if (year == stopyear)
                         {
                              for (int k=1; k< 13; k++)
                             {
                                 for (int m=1; m <= nDaysInMonth(k,
stopyear); m++)
                                 {
                                     if (sundaycounter % 7 == 0 )
                                     {

System.out.println(m+"/"+k+"/"+stopyear+ " Sunday");
                                     }
                                     else
System.out.println(m+"/"+k+"/"+stopyear);
                                     sundaycounter++;
                                 }

                             }

                         }
                    else if (year == stopyear + 1)
                         stop= false;

                         else
                         {
                              i = 1;
                             j = 0;
                         }

                    }
             }

                }
```

```
        }
    }

    private static boolean isLeapYear(int year) {
          if(year % 4 == 0 )
          {
           if (year % 100 == 0)
             {
               if (year % 400 == 0)
               {
                    return true;
               }
               else return false;
             }
             else return true;
          }
          else return false;
     }

     private static int nDaysInMonth(int month, int year) {
          if (month == 4 || month == 6 || month == 9 || month ==
11)
          {
          return 30;
          }
          else if (month == 1 || month == 3 || month == 5 || month
== 7 || month == 8 || month == 10 || month == 12)
          {
               return 31;
          }
          else if (isLeapYear(year))
          {
               if (month == 2)
               return 29;
          }
          return 28;
     }

}
```

```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) {
        iterationCounter = 0;
        double g = loan/n;
        while (endBalance(loan, rate, n, g) >= 0) {
            if (endBalance(loan, rate, n, g) < 0) {
                break;
            }
            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 l = loan/n;
        double h = loan;
        double g = (l+h) / 2;
        iterationCounter = 0;
        while ((h-l)> epsilon)
        {
             iterationCounter++;
            if (endBalance(loan, rate, n, g)*endBalance(loan,
rate, n, l) > 0)
            {
                l = g;
            }
            else h = g;
            g= (l+h)/2;
        }
        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) {
        for(int i = 0; i < n; i++)
        {
            loan = (loan-payment)*(1+rate/100);
        }
        return loan;
    }
}
```

```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 news = "";
        char ch = ' ';
        for (int i = 0; i < s.length(); i++)
        {
           ch = s.charAt(i);
           if ((s.charAt(i) >= 'A') && (s.charAt(i) <= 'Z')) //
check if the letter if the letter is capital char
            {
              char lowercaseChar = (char) (ch + ('a' - 'A')); // c
= 'B'+32; then c stores 'b'
              news += lowercaseChar;
            }
            else news += ch;

        }
        return news;
    }
}
```

```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  news = "";
        news+= s.charAt(0);
        for(int i=0;i <s.length();i++)
        {
            char ch = s.charAt(i);
            if (ch== ' ')
            news+=' ';
            else if(news.indexOf(ch) == -1)
            {
                news += ch;
            }

        }
        return news;
    }
}
```