```
/**
 * 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 calendars of all the years in the 20th century. Also prints the
    * number of Sundays that occured on the first day of the month during this period.
    */
   public static void main(String args[]) {
      // Advances the date and the day-of-the-week from 1/1/1900 till 31/12/1999, inclusive.
      // Prints each date dd/mm/yyyy in a separate line. If the day is a Sunday, prints "Sunday".
      // The following variable, used for debugging purposes, counts how many days were advanced
so far.
      int debugDaysCounter = 0;
      int total_sunday_first=0;
      int yearr = Integer.parseInt(args[0]);
      //// Write the necessary initialization code, and replace the condition
      //// of the while loop with the necessary condition
      while (year<=yearr) {
         //// Write the body of the while
         debugDaysCounter++;
         if(dayOfWeek==1 && year==yearr){
            if(dayOfMonth==1){
               total_sunday_first++;
            }
            System.out.println(dayOfMonth +"/"+ month +"/"+ year + " Sunday");
         }
         else if(dayOfWeek!=1 && year==yearr){
            System.out.println(dayOfMonth +"/"+ month +"/"+ year);
         }
         advance();
         //// If you want to stop the loop after n days, replace the condition of the
         //// if statement with the condition (debugDaysCounter == n)
      }
      System.out.println("During the 20th century, "+ total_sunday_first + " Sundays fell on the first
day of the month");
```

```
    //// Write the necessary ending code here
}

// 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 (dayOfMonth==nDaysInMonth){
        if (month==12){
            dayOfMonth=1;
            month=1;
            year++;}
        else{
            month+=1;
            dayOfMonth=1;}
    }
    else{
        dayOfMonth++;
    }
    dayOfWeek=(dayOfWeek%7)+1;
}



// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        return true;
    } else {
        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) {
    if(month==4 || month==6 || month==9 || month== 11){
        return 30;
    }
    else if(month==2) {
```

```
            if(isLeapYear(year)){
                return 29;
            }
            else{
                return 28;
            }
        }
        else{
            return 31;
        }
    }
}


 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar1 {
    // 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 calendars of all the years in the 20th century. Also prints the
* number of Sundays that occured on the first day of the month during this period.
*/
public static void main(String args[]) {
// Advances the date and the day-of-the-week from 1/1/1900 till 31/12/1999, inclusive.
   // Prints each date dd/mm/yyyy in a separate line. If the day is a Sunday, prints "Sunday".
   // The following variable, used for debugging purposes, counts how many days were advanced so
far.
   int debugDaysCounter = 0;
int total_sunday_first=0;
   //// Write the necessary initialization code, and replace the condition
   //// of the while loop with the necessary condition
while (year<=1999) {
//// Write the body of the while
debugDaysCounter++;
if(dayOfWeek==1){
if(dayOfMonth==1){
total_sunday_first++;
}
```

```
System.out.println(dayOfMonth +"/"+ month +"/"+ year + " Sunday");
}
else{
System.out.println(dayOfMonth +"/"+ month +"/"+ year);
}
advance();
//// If you want to stop the loop after n days, replace the condition of the
//// if statement with the condition (debugDaysCounter == n)
      }
System.out.println("During the 20th century, "+ total_sunday_first + " Sundays fell on the first day of
the month");


//// Write the necessary ending code here
}

// 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 (dayOfMonth==nDaysInMonth){
if (month==12){
dayOfMonth=1;
month=1;
year++;}
else{
month+=1;
dayOfMonth=1;}
}
else{
dayOfMonth++;
}
dayOfWeek=(dayOfWeek%7)+1;
}



   // Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
return true;
} else {
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) {
if(month==4 || month==6 || month==9 || month== 11){
return 30;
}
else if(month==2) {
if(isLeapYear(year)){
return 29;
}
else{
return 28;
}
}
else{
return 31;
}
}
}

/**
* 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=0;    // 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 payment=loan/n;
while (endBalance(loan, rate, n, payment)> 0) {
payment+=epsilon;
iterationCounter++;
}
    // Replace the following statement with your code
    return payment;
    }

    /**
* 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 paymentL = 0;
double paymentR = loan;

while (paymentR - paymentL > epsilon) {
double mid = (paymentR + paymentL) / 2.0;
```

```
if (endBalance(loan, rate, n, paymentL) * endBalance(loan, rate, n, mid) > 0) {
paymentL = mid;
iterationCounter++;
} else {
paymentR = mid;
iterationCounter++;
}
}

return paymentL;
}


/**
* 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));
}
// Replace the following statement with your code
    return loan;
}
}
/** 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 = "";
        int i = 0;
        while (i < s.length()) {
            char ch = s.charAt(i);
            if (ch >= 'A' && ch <= 'Z') {
                ans = ans + (char) (ch + 32);
```

```
            } else {
                ans = ans + ch;
            }
            i++;
        }
        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 = "" + s.charAt(0) ;
        int i = 1;
        while(i<s.length()){
            char c = s.charAt(i);
            if (i == (s.indexOf(c)) || c==32){
                ans = ans + c;
            }
            else{
                ans = ans + "";
            }
            i++;
        }

        // Replace the following statement with your code
        return ans;
    }

}
```