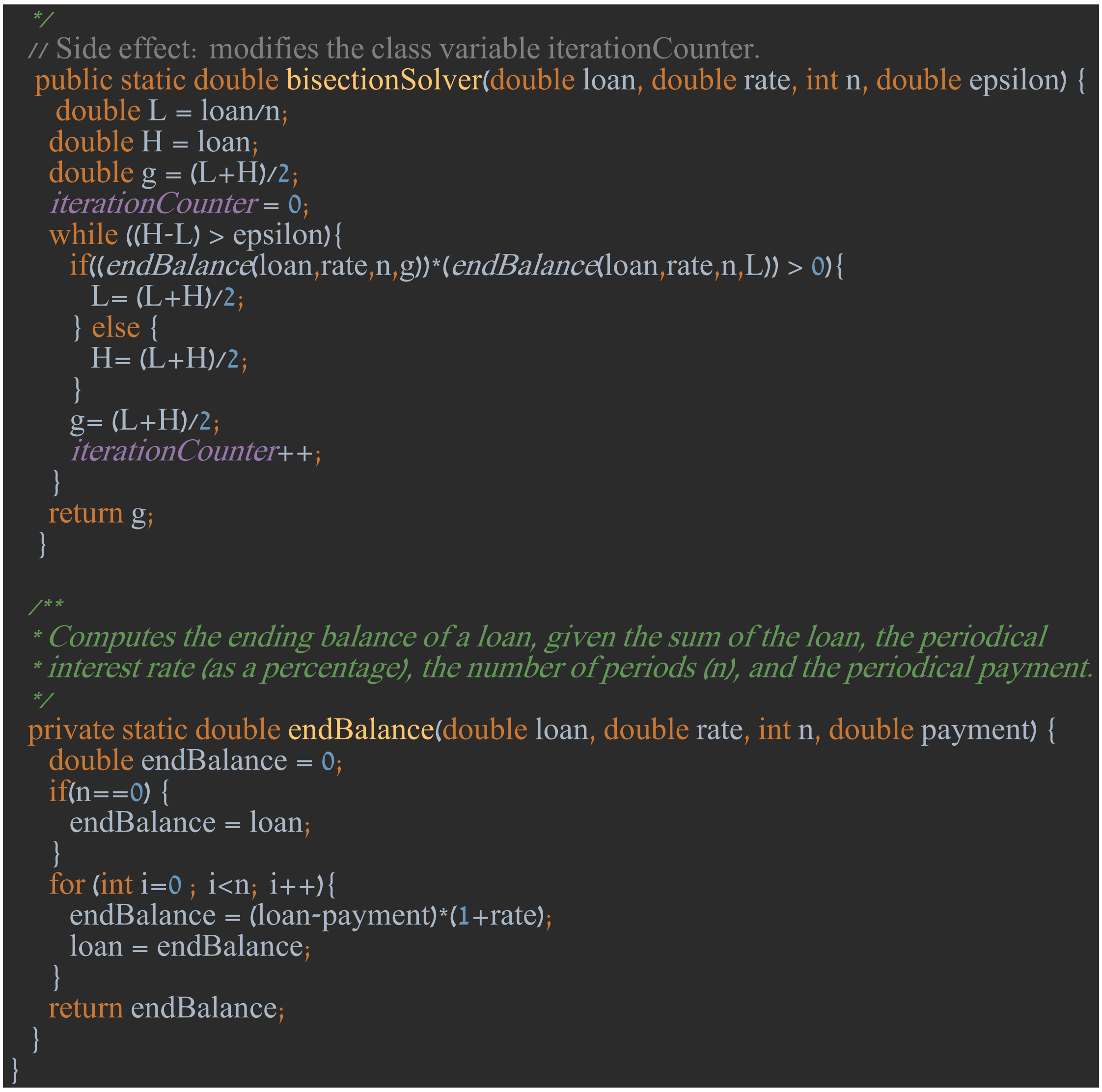
HW3

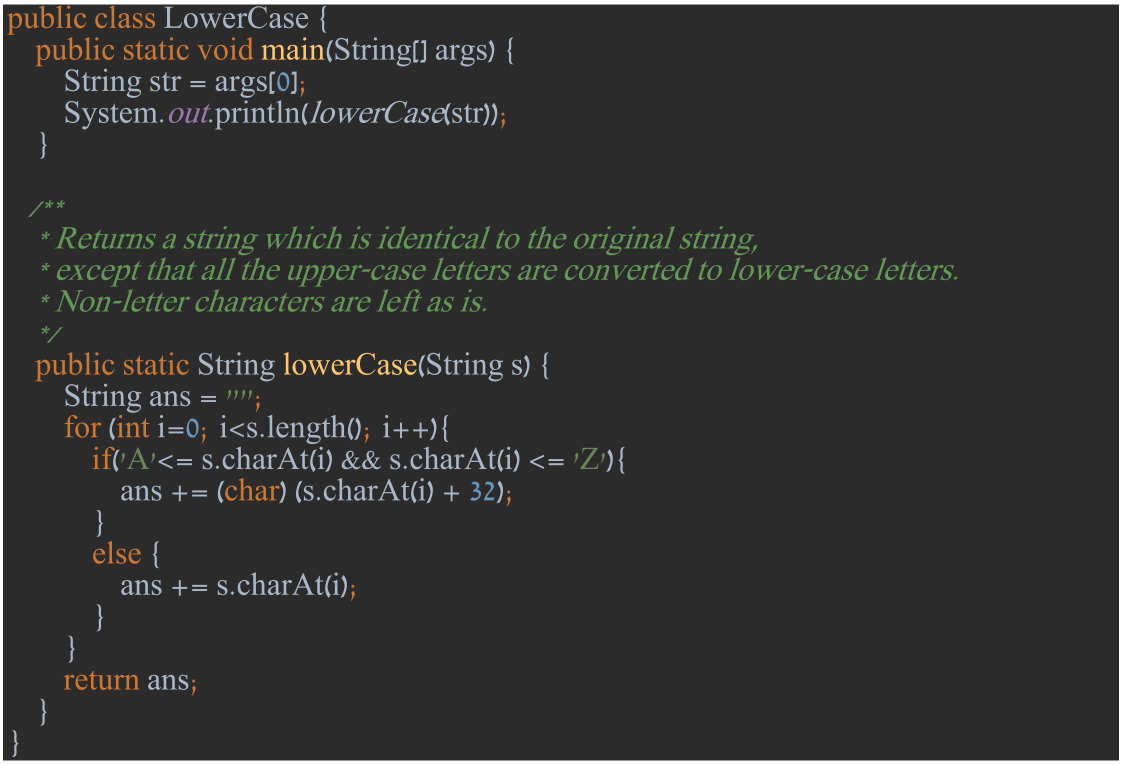
Loan.calc

```
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;
    iterationCounter = 0;
    while (endBalance(loan,rate,n,g) > 0) {
      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 the loan, the periodical interest rate (as a percentage),
  * the number of periods (n), and epsilon, a tolerance level.
```



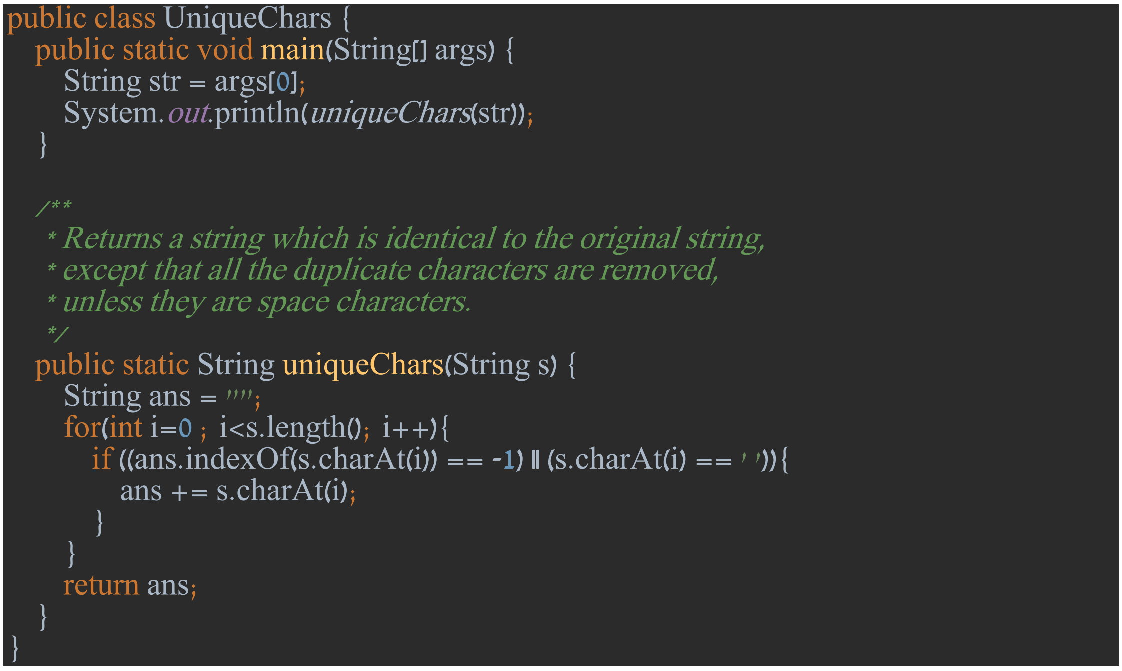
```
   */
  // Side effect: modifies the class variable iterationCounter.
   public static double bisectionSolver(double loan, double rate, int n, double epsilon) {
      double L = loan/n;
     double H = loan;
     double g = (L+H)/2;
     iterationCounter = 0;
     while ((H-L) > epsilon){
       if((endBalance(loan,rate,n,g))*(endBalance(loan,rate,n,L)) > 0){
         L= (L+H)/2;
       } else {
         H= (L+H)/2;
       }
       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 endBalance = 0;
     if(n==0) {
       endBalance = loan;
     }
     for (int i=0 ; i<n; i++){
       endBalance = (loan-payment)*(1+rate);
       loan = endBalance;
     }
     return endBalance;
   }
}
```

LowerCase



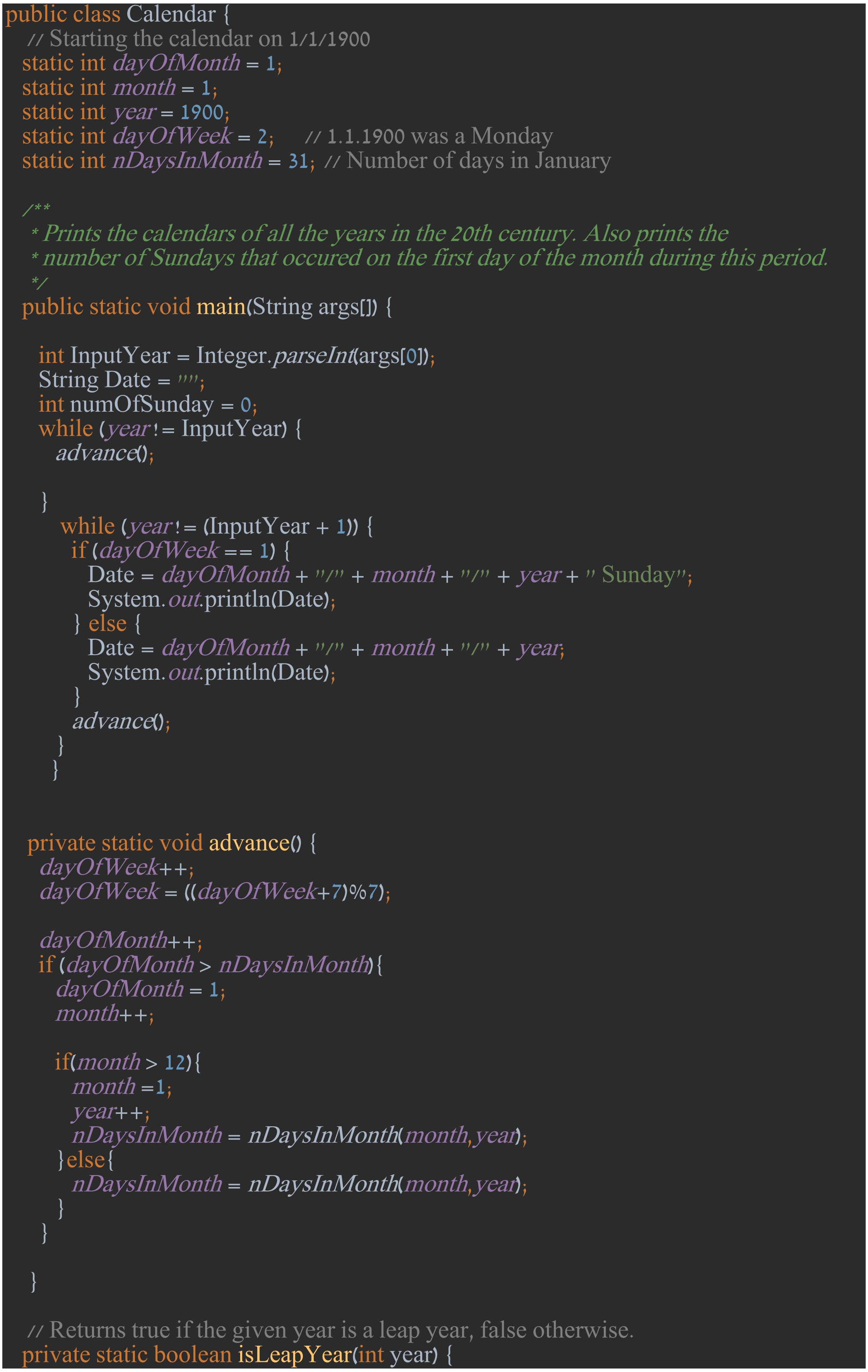
```
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

   /**
    * 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++){
            if('A'<= s.charAt(i) && s.charAt(i) <= 'Z'){
                ans += (char) (s.charAt(i) + 32);
            }
            else {
                ans += s.charAt(i);
            }
        }
        return ans;
    }
}
```

UniqueChars

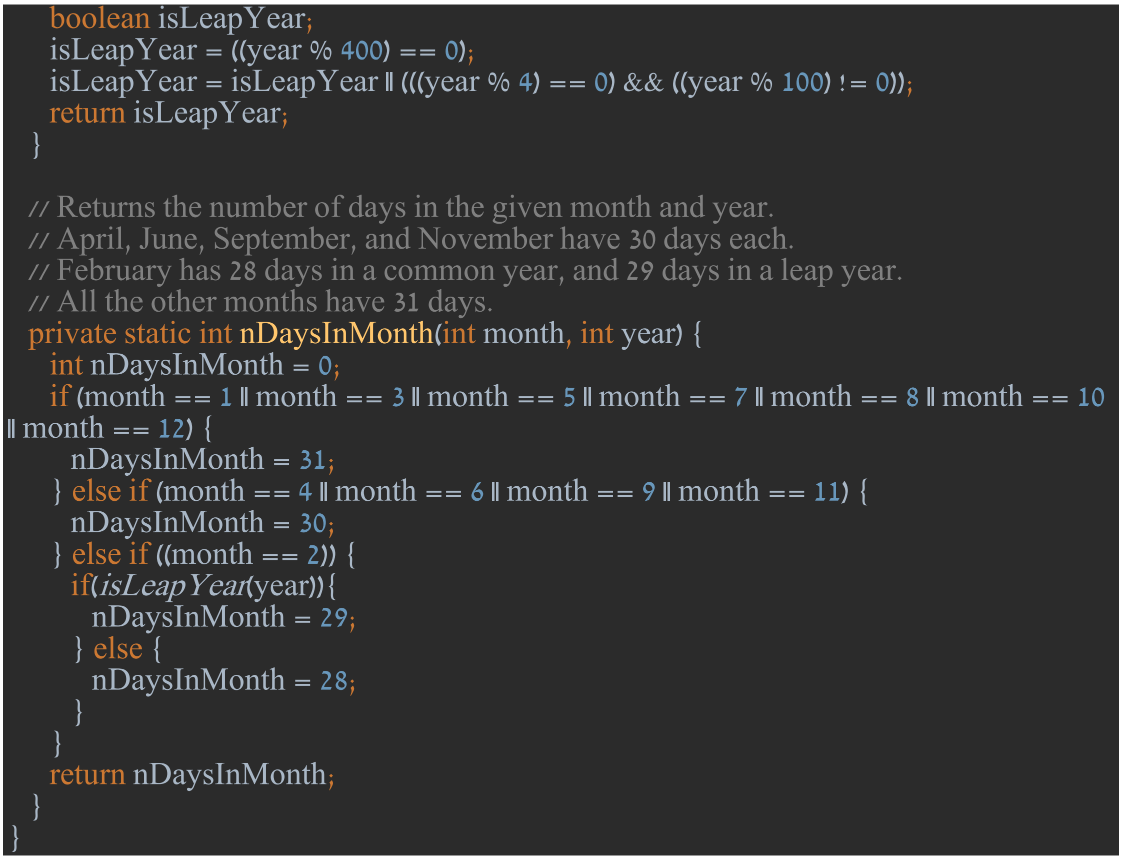


```java
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    /**
     * 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 = "";
        for(int i=0 ; i<s.length(); i++){
            if ((ans.indexOf(s.charAt(i)) == -1) || (s.charAt(i) == ' ')){
                ans += s.charAt(i);
            }
        }
        return ans;
    }
}
```

Calendar



```
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[]) {

    int InputYear = Integer.parseInt(args[0]);
    String Date = "";
    int numOfSunday = 0;
    while (year != InputYear) {
      advance();

    }
       while (year != (InputYear + 1)) {
        if (dayOfWeek == 1) {
          Date = dayOfMonth + "/" + month + "/" + year + " Sunday";
          System.out.println(Date);
        } else {
          Date = dayOfMonth + "/" + month + "/" + year;
          System.out.println(Date);
        }
        advance();
      }
      }


   private static void advance() {
    dayOfWeek++;
    dayOfWeek = ((dayOfWeek+7)%7);

    dayOfMonth++;
    if (dayOfMonth > nDaysInMonth){
      dayOfMonth = 1;
      month++;

      if(month > 12){
        month =1;
        year++;
        nDaysInMonth = nDaysInMonth(month,year);
      }else{
        nDaysInMonth = nDaysInMonth(month,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);
    isLeapYear = isLeapYear || (((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 nDaysInMonth = 0;
    if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10
|| month == 12) {
      nDaysInMonth = 31;
    } else if (month == 4 || month == 6 || month == 9 || month == 11) {
      nDaysInMonth = 30;
    } else if ((month == 2)) {
      if(isLeapYear(year)){
        nDaysInMonth = 29;
      } else {
        nDaysInMonth = 28;
      }
    }
    return nDaysInMonth;
  }
}
```