HW3
1.LoanCalc.java

```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) {
double guess = loan / n;
double increment = 0.001;
        double balance = endBalance( loan , rate , n , guess ) ;
        iterationCounter = 0 ;
        while (balance >= epsilon && balance >= 0) {
                guess += increment ;
                balance = endBalance( loan , rate , n , guess ) ;
                iterationCounter++;
        }
        return guess ;
}

/**
        * 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 guess = ( l + h ) / 2 ;
        iterationCounter = 0;
        while ( (h - l) > epsilon)
        {
                if ( endBalance(loan , rate , n , guess) * endBalance(loan , rate , n , l) > 0 ) {
                        l = guess ;
                }
                else {
                        h = guess ;
                }
                guess = (l + h) / 2 ;
                iterationCounter++;
        }
        return guess ;
}

        /**
        * 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 balance = loan ;
                    for (int i = 0 ; i < n ; i++) {
                              balance = ( balance - payment ) * (1 + 0.01 * rate) ;
                    }
          return balance;
          }
}
```

2.LowerCase

```
/** 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 = "" ;
        int i = 0 ;
        while ( i < s.length ()) {
        char c = s.charAt(i) ;
           if (c >= 65 && c <= 90) {
               c = (char)(c + 32) ;
           }
           newS += c ;
            i = i + 1;
        }
        return newS;
        }
}
```

3.UniqueChars.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 = "" ;
      for (int i = 0 ; i < s.length() ; i++) {
         boolean exist = false ;
         char c = s.charAt(i);
         if (c != 32) {
           for (int j = 0 ; j < newS.length() ; j++) {
            if ( newS.charAt(j) == c )
              exist = true ;
             }
         }
         if (exist == false )
         {
            newS = newS + c;
         }
   }
   return newS;
}}
```

4a.Calendar0.java

```
/*
 * Checks if a given year is a leap year or a common year,
 * and computes the number of days in a given month and a given year.
 */
public class Calendar0 {

	// Gets a year (command-line argument), and tests the functions isLeapYear and 
nDaysInMonth.
	public static void main(String args[]) {
		int year = Integer.parseInt(args[0]);
		isLeapYearTest(year);
		nDaysInMonthTest(year);
	}

	// Tests the isLeapYear function.
	private static void isLeapYearTest(int year) {
		String commonOrLeap = "common";
		if (isLeapYear(year)) {
			commonOrLeap = "leap";
		}
		System.out.println(year + " is a " + commonOrLeap + " year");
	}

	// Tests the nDaysInMonth function.
	private static void nDaysInMonthTest(int year) {
		for (int i=1 ; i <=12 ; i++) {
			int days = nDaysInMonth(i , year);
			System.out.println("Month " + i + " has " + days + " days" ) ;
		}
	}

	// Returns true if the given year is a leap year, false otherwise.
	public 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.
	public static int nDaysInMonth(int month, int year) {
		if ((month == 1) || (month == 3) || (month == 7) || (month == 8) || (month
==10) || (month == 12) || (month == 5)){
			return 31;
		}
		if (month == 2) {
			if (isLeapYear(year)== true) {
				return 29;
			}
			else {
				return 28 ;
			}
		}
		return 30;
	}
}
```

4b.Calendar1.java

```
/**
 * 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 sundays = 0 ;
             //// Write the necessary initialization code, and replace the condition
             //// of the while loop with the necessary condition
                    while ((year != 2000) ||  (month != 1) || (dayOfMonth != 1)) {
                              if (dayOfWeek==1) {
                                        System.out.println(dayOfMonth+"/"+month+"/"+year+" sunday");
                                        if (dayOfMonth==1){
                                                  sundays++ ;
                                        }
                              } else {
                                        System.out.println(dayOfMonth+"/"+month+"/"+year);
                              }
                              advance();
                              }
                              debugDaysCounter++;
                    System.out.println("During the 20th century, " + sundays + " 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() {
                        if (dayOfWeek==7) {
                                    dayOfWeek = 1 ;
                        } else {
                                    dayOfWeek++;
                        }
                        if (dayOfMonth == nDaysInMonth(month,year)) {
                                    dayOfMonth = 1;
                                    if (month == 12) {
                                                month = 1;
                                                year++;
                                    }
                                    else {
                                                month++;
                                    }
                        } else {
                                    dayOfMonth++;
                        }
             }

   // 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) {
                        if ((month == 1) || (month == 3) || (month == 7) || (month == 8) || (month
==10) || (month == 12) || (month == 5)) {
                                    return 31;
                        }
                        if (month == 2) {
                                    if (isLeapYear(year)== true) {
                                                return 29;
                                    }
            else {
                return 28 ;
            }
        }
        return 30;

    }
}
```

4c.Calendar.java

```
/**
 * Prints the calendars of the selected year
 */
public class Calendar {
	static int dayOfMonth = 1;
	static int year = 1990 ;
	static int month = 1;
	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 days in the selected year in the year.
	 */
	public static void main(String args[]) {
		int Myyear = Integer.parseInt(args[0]);
		while ((year != Myyear) ||  (month != 1) || (dayOfMonth != 1)) {
			advance();
			}
		while ((year != Myyear + 1) ||  (month != 1) || (dayOfMonth != 1)) {
			if (dayOfWeek == 1){
				System.out.println(dayOfMonth+"/"+month+"/"+year+" Sunday");
			}
			else {
				System.out.println(dayOfMonth+"/"+month+"/"+year);
			}
			advance();
			}
	}

	private static void advance() {
		if (dayOfWeek==7) {
			dayOfWeek = 1 ;
		} else {
			dayOfWeek++;
		}
		if (dayOfMonth == nDaysInMonth(month,year)) {
			dayOfMonth = 1;
			if (month == 12) {
				month = 1;
				year++;
			}
			else {
				month++;
			}
			}
			else {
				dayOfMonth++;
			}

		 }
   // 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) {
			if ((month == 1) || (month == 3) || (month == 7) || (month == 8) || (month
==10) || (month == 12) || (month == 5)) {
				return 31;
			}
			if (month == 2) {
				if (isLeapYear(year)== true) {
					return 29;
				}
				else {
					return 28 ;
				}
			}
			return 30;

		}
}
```