1) LoanCalc

```
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(endBalance (100000, 5, 3, 10000));
      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) {
         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)
{
        // Replace the following statement with your code
        iterationCounter = 0;

        double L = loan / n;
        double H = loan;
        double g = ((H + L)/2);

        while ((H - L) > epsilon) {
            // Sets L and H for the next iteration
            if (endBalance(loan,rate,n,g)*endBalance(loan,rate,n,L) > 0) {
            // the solution must be between g and H
            // so set L or H accordingly
            L = g;
        }
            else {
            H = g;
        }
        g = (H + L)/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 endB = loan;

        for (int i = 0; i < n; i++) {
            endB = ((endB - payment) * (rate/100 + 1));

        }
        return endB;
```

```
    }
}
```

2) LowerCase

```
public class LowerCase {

    public static void main(String[] args) {
        if (args.length > 0) {
            String str = args[0];
            System.out.println(lowerCase(str));
        } else {
            System.out.println("No string provided!");
        }
    }

    public static String lowerCase(String s) {
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < s.length(); i++) {
            char currentChar = s.charAt(i);
            if (currentChar >= 'A' && currentChar <= 'Z') {
                // Convert uppercase to lowercase by adding 32 (based on ASCII values)
                result.append((char) (currentChar + 32));
            } else {
                // If not uppercase, add the character as is
                result.append(currentChar);
            }
        }

        return result.toString();
    }
}
```

3) UniqueChars

```
public class UniqueChars {
```

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

    public static String uniqueChars(String s) {
        String result = "";
        for (int i = 0; i < s.length(); i++) {
            char currentChar = s.charAt(i);
            if (result.indexOf(currentChar) == -1 || currentChar == ' ') {
                result += currentChar;
            }
        }
        return result;
    }
}
```

4) Calendar0

```
public class Calendar0 {

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Calendar0 <year>");
```

```
            return;
        }

        int year = Integer.parseInt(args[0]);
        isLeapYearTest(year);
        nDaysInMonthTest(year);
    }

    private static void isLeapYearTest(int year) {
        boolean leapYear = isLeapYear(year);
        if (leapYear) {
            System.out.println(year + " is a leap year");
        } else {
            System.out.println(year + " is a common year");
        }
    }

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

    public static boolean isLeapYear(int year) {
        return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
    }

    public static int nDaysInMonth(int month, int year) {
        switch (month) {
            case 2:
                return isLeapYear(year) ? 29 : 28;
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            default:
                return 31;
        }
    }
}
```

5) Calendar1

```
public class Calendar1 {
        // Starting the calendar on 1/1/1900
        static int curMonth;
        static int curDay;
        static int curYear;
        static int endYear;
        static int curDayOfWeek; // ==> (2) 1.1.1900 was a Monday
        static int nDaysInMonth; // num of days at curr month
        static boolean isLeapYear; // true if year is a leap year
        static int nDays; // number of days in the month
        static int countSunday;
```

```
	public static void main(String args[]) {
		advance();
	}

	/**
	 * This founction print the calender from 1990 - 1999 inclusive.
	 */
	public static void advance() {
		curYear = 1900;
		endYear = 1999;
		curDayOfWeek = 2;
		countSunday = 0;
		while (curYear <= endYear) {
			curMonth = 1;
			while (curMonth <= 12) {
				curDay = 1;
				while (curDay <= nDaysInMonth(curMonth, curYear)) {
					if (curDayOfWeek <= 7) {
						System.out.print(curDay + "/" + curMonth +
"/" + curYear);
						if ((curDay == 1) && (curDayOfWeek) == 1)
{
							System.out.print(" Sunday");
							countSunday++;
							curDay++;
							curDayOfWeek++;
						} else {
							curDay++;
							curDayOfWeek++;
						}
						if (curDayOfWeek > 7) {
							curDayOfWeek = 1;
						}
					}
					System.out.println();
				}
				curMonth++;
			}
			curYear++;
		}
		System.out.println("During the 20th century, " + countSunday + "
Sundays fell on the first day of the month");
	}

	/**
	 * This founction return if the year us leap or common.
	 *
	 * @param year - represents the year
	 * @return - true if the given year is a leap year, false otherwise.
```

```
	 */
	private static boolean isLeapYear(int year) {
		// check if the year is divisble by 400
		isLeapYear = ((year % 400) == 0);
		// than checks if the year is divisible by 4 and not by 100
		isLeapYear = isLeapYear || ((year % 4) == 0 && (year % 100) != 0);
		return isLeapYear;
	}

	/**
	 * Returns the number of days in the given month and year. April, June,
	 *
	 * @param month - represents the month
	 * @param year  - represents the year
	 * @return - the number of days in the given month and year
	 */
	private static int nDaysInMonth(int curMonth, int curYear) {
		switch (curMonth) {
			case 1, 3, 5, 7, 8, 10, 12: // January, March, May, July, August,
October, and December
				nDays = 31;
				break;
			case 2: // February
				nDays = isLeapYear(curYear) ? 29 : 28;
				break;
			case 4, 6, 9, 11: // April, June, September, and November
				nDays = 30;
				break;
			default:
				nDays = 0;
				System.out.println("Invalid month");
				break;
		}
		return nDays;
	}
}
```

6) 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 checkyear= Integer.parseInt(args [0]);
      int sundayCounter=0;
                  while (year<checkyear)
       {
```

```
          advance();
        }
        while    (year==checkyear)
        {
          System.out.print(dayOfMonth+"/"+month+"/"+year);
          if(dayOfWeek==1)
                        {
                                System.out.print(" Sunday");
                        }
                        System.out.println();
                        advance();

      }
   }



        // 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(month==12 && dayOfMonth==31)// if you made it to the end of the
year start a new one
                {
                        year++;
                        month=1;
                        dayOfMonth=1;
                }
                else
                {

                        if (dayOfMonth==nDaysInMonth)//if you made it to the end of the
month start a new one, else advance
                        {
                                month++;
                                if (month==13)
                                        month=1;
                                nDaysInMonth=nDaysInMonth(month, year);
                                dayOfMonth=1;

                        }
                        else
                                dayOfMonth++;
                }
                if(dayOfWeek==7)//if you made it to the end of the week, start over
                        dayOfWeek=1;
                else
                        dayOfWeek++;
```

```
        }

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

        private static int nDaysInMonth(int month, int year)
         {
                int days = 31;
                switch (month)
                {
                        case 4: days= 30 ;
                        break;
                        case 6: days= 30 ;
                        break;
                        case 9: days= 30 ;
                        break;
                        case 11: days= 30 ;
                        break;
                        case 2:
                                if (isLeapYear(year))
                                        days=29;
                                else
                                        days=28;
                                break;
                        }
                return days;
        }
}
```