**Program Projects 2 Some Additional Useful Types and Classes and Array Lists – Spring 2021**

Notes: All homework must be submitted via e-mail. All parts of assignment must be submitted in a single e-mail with multiple attachments when required.

Notes: All homework must be submitted via e-mail. All parts of assignment must be submitted in a single e-mail with multiple attachments.

E-mail address is:

[csc123csudh@gmail.com](mailto:csc123csudh@gmail.com)

Each program is to be submitted in a separate file with the file name being the class name with extension .java as shown below. I only need the source file.

**GeneralAverage.java**

All answers will be posted on web site, and most will be reviewed in class.

1. **SalesAnalysis (40)**

Write a program SalesAnalysis That does the following. You can use the salesdata.txt file in the Program Projects Answers/Project 2 tab for input.

The data file will have a set of comma delimited values of type double

In **main**:

Create a Scanner to read input from the keyboard.

Request the name of the input file to read data from.

While there is data in the file

Read in a line of data as a String

Increment the week number, which was initialized to 0.

Call getWeeklySales with the input line, returning the sum of the numbers as a double

Print out the total sales for the week.

Print out the average for the week (assume that there are sales all seven days)

Accumulate the total weekly sales

Determine highest to lowest so far

Close the Scanner that read in the data

Print out the additional data as shown in the Sample output below.

**getWeeklySales** method

Accepts a String with 7 decimal numbers delimited by a comma

Trim the white space

Tokenize the String in an array of type String

Sum together the elements using the Double.parseDouble method.

Return the sum.

Sample input/output:

Please enter the name of your input file: SalesData.txt

Total sales for week 1: $12,092.75

Average daily sales for week 1: $1,727.54

Total sales for week 2: $27,461.00

Average daily sales for week 2: $3,923.00

Total sales for week 3: $12,058.34

Average daily sales for week 3: $1,722.62

Total sales for all weeks: $51,612.09

Average weekly sales: $17,204.03

The highest sales were made during week 2.

The lowest sales were made during week 3.

1. **ISBNVerifier (50)**

Background description:

An International Standard Book Number (ISBN) is a code that uniquely identifies an edition of a book. The ISBN consists of ten digits separated by dashes into groups. The groups are of various sizes, except for the last group. The last group is always a single character, '0' through '9' or 'X', and acts as a check on the rest of the digits. In this problem we look at 10 digit ISBN’s

0-670-03441-X

0-201-48558-3

1-56592-262-X

0-06-027900-1

0-439-45695-9

0-470-84371-3

1-4000-3136-2

0-19-856453-8

1-85671-104-8

The last character is calculated from the other 9 digits:

Key Algorithm to calculate check digit.

Multiply the first digit by 10.

Multiply the second digit by 9.

Multiply the second digit by 8.

. . .

Multiply the ninth digit by 2.

Add up all these values.

Integer divide the sum by 11.

Find the remainder.

Subtract the remainder from 11. This is the check digit.

If the checkDigit is 10, use an 'X'

For example:

0-201-48558-3

0 \* 10 = 0

2 \* 9 = 18

0 \* 8 = 0

1 \* 7 = 7

4 \* 6 = 24

8 \* 5 = 40

5 \* 4 = 20

5 \* 3 = 15

8 \* 2 = 16

-----

sum = 140

140 / 11 = 12 rem 8

11 - 8 = 3 <-- the check digit

Write a program that prompts the user for a file containing ISBN numbers and then checks if each the ISBN is correct. I have posted the isbnCodes.txt file in the Program Projects Answers/Project 2 tab on the web site and you may use it to test your results

Here is a complete description of what I want you to do:

In **main**:

Create a Scanner to read in the name of the input file as a String.

Call a method **createScanner** with the parameter of the String file. **createScanner** should return a reference to a scanner that allows you to read from the input file.

For each ISBN code in the file

Read in the potential ISBN codes one at a time as a StringBuilder.

Call checkISBN with the reference to StringBuilder object you have read in. **checkISBN** returns a String reference with a message telling whether this is a valid code, and if not why not.

Print the message to the screen , along with the ISBN number entered. (see sample output when using the file I have provided).

Close the Scanner used to read he file.

In **createScanner**:

Accept the reference to the file name as a String.

Create a Scanner object that reads from that file. Be sure to check for the file’s existence.

Return a reference to the created Scanner

In **checkISBN**:

Perform the following checks. If a check leads to an error, return a message listing the error. Below is my suggested order.

Here are steps you may follow:

Check that total length is 13.

Make sure the 12th position is a digit or X

Make sure that there are exactly three ‘-‘ in the code.

Check to make sure that each character except the last is a digit or a dash.

Extract the nine digits in order

Calculate the check digit.

If the check digit matches as described above return a message indicating a good ISBN, otherwise return a message indicating that a bad ISBN was created because the digit doesn’t match

**Helpful Hint:**

Here is an example of how to convert the char to an int:

char x = '9';

int y = x - '0'; // gives 9 remember the subtraction will cast up to int and the int characters are sequential in UNICODE.

Sample output:

Please enter a the file name containing the ISBN Files: isbnCodes.txt

A check on the ISBN 0-670-03441-X yields: Good - final digit is X

A check on the ISBN 0-201-48558-3 yields: Good Digits Match

A check on the ISBN 1-56592-262-X yields: Good - final digit is X

A check on the ISBN 0-06-027900-1 yields: Good Digits Match

A check on the ISBN 0-439-45695-9 yields: Good Digits Match

A check on the ISBN 0-470-84371-3 yields: Good Digits Match

A check on the ISBN 1-4000-3136-2 yields: Good Digits Match

A check on the ISBN 0-19-856453-8 yields: Good Digits Match

A check on the ISBN 1-85671-104-8 yields: Good Digits Match

A check on the ISBN 92-9395 yields: Bad Length

A check on the ISBN 42-4920-223-Y yields: Final Character is Bad

A check on the ISBN 1-8906-456789 yields: There are not 3 dashes as required

A check on the ISBN 1-856X1-104-8 yields: Character not a digit or -

A check on the ISBN 1-4000-3136-5 yields: Final digit doesn't match check digit or X

A check on the ISBN 0-670-03441-X yields:Good - final digit is X

A check on the ISBN 0-201-48558-3 yields:Good Digits Match

A check on the ISBN 1-56592-262-X yields:Good - final digit is X

A check on the ISBN 0-06-027900-1 yields:Good Digits Match

A check on the ISBN 0-439-45695-9 yields:Good Digits Match

A check on the ISBN 0-470-84371-3 yields:Good Digits Match

A check on the ISBN 1-4000-3136-2 yields:Good Digits Match

A check on the ISBN 0-19-856453-8 yields:Good Digits Match

A check on the ISBN 1-85671-104-8 yields:Good Digits Match

A check on the ISBN 92-9395 yields:Bad Length

A check on the ISBN 42-4920-223-Y yields:Final Character is bad

A check on the ISBN 1-8906-456789 yields:There are not 3 dashes as required

A check on the ISBN 1-856X1-104-8 yields:Good Digits Match

A check on the ISBN 1-4000-3136-5 yields:Final digit doesn't match check digit or X

1. **CompareTimes (20)**

This problem can be done in main. Apply the increment operator (--) 10 million times, first to an Integer reference and then to a variable of type int, each set initially to -1. Compare the running times.

To compare running times we use the method **long System.currentTimeMillis()**

that returns the current time in milliseconds.

For both the Integer reference x and the primitive int y, the main method

* records the starting time in milliseconds,
* decrements with -- a variable 10,000,000 times,
* records the ending time in milliseconds, and
* displays the elapsed time, *ending time* - *starting time*.

Sample Output:

Wrapper time: 67 milliseconds

integer time: 1 milliseconds

1. **SieveOfPrimes (30)**

Write a program that creates a Boolean ArrayList with a capacity of 1000.

Initialize each value in the ArrayList to true.

For each index in the ArrayList, if the index is not a prime number set the value of the ArrayList at that index to false.

When you are finished create a loop that prints the following output followed by the total number of primes from 0 to 999.

You can use the previous mechanisms for calculating a prime or the algorithm described below which is known as the Sieve of Eratosthenes. Either way the output should look the same.

Starting with array index 2, determine whether a given element is true. If so, loop through the remainder of the array and set to false every element whose index is a multiple of the index for the element with value true. Then continue the process with the next element with value true. For array index 2, all elements beyond element 2 in the array that have indices which are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to false; for array index 3, all elements beyond element 3 in the array that have indices which are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to false; and so on.

2 is prime.

3 is prime.

5 is prime.

7 is prime.

11 is prime.

13 is prime.

17 is prime.

19 is prime.

23 is prime.

29 is prime.

31 is prime.

37 is prime.

41 is prime.

43 is prime.

47 is prime.

53 is prime.

59 is prime.

61 is prime.

67 is prime.

71 is prime.

73 is prime.

79 is prime.

83 is prime.

89 is prime.

97 is prime.

101 is prime.

103 is prime.

107 is prime.

109 is prime.

113 is prime.

127 is prime.

131 is prime.

137 is prime.

139 is prime.

149 is prime.

151 is prime.

157 is prime.

163 is prime.

167 is prime.

173 is prime.

179 is prime.

181 is prime.

191 is prime.

193 is prime.

197 is prime.

199 is prime.

211 is prime.

223 is prime.

227 is prime.

229 is prime.

233 is prime.

239 is prime.

241 is prime.

251 is prime.

257 is prime.

263 is prime.

269 is prime.

271 is prime.

277 is prime.

281 is prime.

283 is prime.

293 is prime.

307 is prime.

311 is prime.

313 is prime.

317 is prime.

331 is prime.

337 is prime.

347 is prime.

349 is prime.

353 is prime.

359 is prime.

367 is prime.

373 is prime.

379 is prime.

383 is prime.

389 is prime.

397 is prime.

401 is prime.

409 is prime.

419 is prime.

421 is prime.

431 is prime.

433 is prime.

439 is prime.

443 is prime.

449 is prime.

457 is prime.

461 is prime.

463 is prime.

467 is prime.

479 is prime.

487 is prime.

491 is prime.

499 is prime.

503 is prime.

509 is prime.

521 is prime.

523 is prime.

541 is prime.

547 is prime.

557 is prime.

563 is prime.

569 is prime.

571 is prime.

577 is prime.

587 is prime.

593 is prime.

599 is prime.

601 is prime.

607 is prime.

613 is prime.

617 is prime.

619 is prime.

631 is prime.

641 is prime.

643 is prime.

647 is prime.

653 is prime.

659 is prime.

661 is prime.

673 is prime.

677 is prime.

683 is prime.

691 is prime.

701 is prime.

709 is prime.

719 is prime.

727 is prime.

733 is prime.

739 is prime.

743 is prime.

751 is prime.

757 is prime.

761 is prime.

769 is prime.

773 is prime.

787 is prime.

797 is prime.

809 is prime.

811 is prime.

821 is prime.

823 is prime.

827 is prime.

829 is prime.

839 is prime.

853 is prime.

857 is prime.

859 is prime.

863 is prime.

877 is prime.

881 is prime.

883 is prime.

887 is prime.

907 is prime.

911 is prime.

919 is prime.

929 is prime.

937 is prime.

941 is prime.

947 is prime.

953 is prime.

967 is prime.

971 is prime.

977 is prime.

983 is prime.

991 is prime.

997 is prime.

168 primes found.