**SCANNER**

**LAB 6.1**

*Write a program, where system will take input values from console, system will ask for user’s birth place, user’s full name and age (one by one) and receive the user inputs using various scanner methods like next(), nextLine() and nextInt().*

*This program claims to tell the real age of user; it counts the length (number of character) in user’s birth place (say x) and adds this to user’s age and will display it as a message below -*

*“Hi USERNAME, you are from USER’S BIRTH PLACE and your actual age is (USER’S AGE + X)….. Ha ha ha”*

**Steps:**

* Create a ScannerPractice class having p.s.v.main method
* Create scanner object and take input from user and display desired output

**package** cg.javaflp.practice;

**import** java.util.Scanner;

**public** **class** ScannerPractice {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.*in*);

// receiving user's name

System.*out*.println("Please enter your name: ");

String name = sc.next();

// receiving user's birth place (name of city)

System.*out*.println("Please enter your Birth place: ");

String birthPlace = sc.next();

// receiving user's age

System.*out*.println("Please enter your age: ");

**int** age = sc.nextInt();

// Counting characters of birth place (city) name

**int** length = birthPlace.length();

age = age + length; // modified the age

// displaying output massage

System.*out*.println("Hi, " + name + " your actual age is "

+ age + "\n" + "MUAHAHAHHAHA");

}

}

**OUTPUT**

Please enter your name: vivek

Please enter your Birth place: pune

Please enter your age: 31

Hi, vivek your actual age is 35

MUAHAHAHHAHA

**Learning:**

* How to use Scanner
* Some important method of scanner

**LAB 6.2**

*Write a program where user will enter four numerical values (one by one) from consol (Using Scanner), and output will be sorted list of these value (ASC and DESC)*

**Steps:**

* Here we simply modified the SortTheNumber class, that we created for java Lang assignment; here instead of raking runtime argument from command line, we are receiving it from console as user input using scanner

**package** cg.javaflp.practice;

**import** java.util.Scanner;

/\*\*

\* This class is use to sort the numbers in ascending and descending order

\* Numbers will be pass on runtime using scanner

\*/

**public** **class** SortTheNumbers {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Please enter four numbers one by one, " +

"press enter key after every number");

// numbers as runtime argument using scanner

Double a = sc.nextDouble();

Double b = sc.nextDouble();

Double c = sc.nextDouble();

Double d = sc.nextDouble();

System.*out*.println("Four numbers to be sorted..");

System.*out*.println(a + ", " + b + ", " + c +", " + d);

/\* Logic to sort many numbers

\* 1. store the numbers as an array

\* 2. compare each element (number) to its next element (number)

\* 3. replace the numbers if first no is greater then second

\* 4. after step 3, array will be modified

\* 5. iterate step 2 and 3, "length of array" times

\*/

// step 1 of above mentioned logic

Double arr[] = **new** Double[]{a, b, c, d};

// outer FOR loop is for step 5

**for** (**int** i = 0; i < arr.length; i++){

// inner FOR loop is for steps 2, 3 and 4

**for** (**int** j = 1 ; j <arr.length; j++){

Double temp;

**if** (arr[j - 1].compareTo(arr[j])>0){

temp = arr[j - 1];

arr[j-1] = arr[j];

arr[j] = temp;

}

}

}

System.*out*.println("Sorted value is : ");

System.*out*.println(arr[0]+", "+arr[1]+", "+arr[2]+", "+arr[3]);

}

}

**OUTPUT**

Please enter four numbers one by one, press enter key after every number

24

57

2

567

Four numbers to be sorted..

24.0, 57.0, 2.0, 567.0

Sorted value is:

2.0, 24.0, 57.0, 567.0

**LAB 6.3**

*Create a calculator, where user will select the arithmetic operation, then user will ask for values on which the operation is required to perform. Now user will provide number and operation will be performed.*

*Write this program with the help of scanner and switch; also for mathematic operation use the previously written MyMath class.*

**Steps:**

* Create Calculator class, having p.s.v.main method
* Using scanner, Take user input for desired operation (1 for add, 2 for subs, 3 for multiply and 4 for divide); also receive two numbers on which user want to perform the operation
* Instantiate MyMath class object (this class have methods for add, subs, multiple, divide, etc)
* Use switch to call appropriate method and show result

**package** cg.javaflp.practice;

**import** java.util.Scanner;

**public** **class** Calculator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Select your mathematical operation: enter "

+ " \n 1 to Add, \n 2 to Substract \n "

+ "3 to Multuply, 4 to Divide");

**int** userChoice = sc.nextInt();

System.*out*.println("Provide two numbers; " +

"press enter/space key after each nos");

**double** firstNumber = sc.nextDouble();

**double** secondNumber = sc.nextDouble();

MyMath myMath = **new** MyMath();

**double** result = 0.0;

**switch** (userChoice) {

**case** 1:

System.*out*.println("Addition of " + firstNumber + " and "

+ secondNumber + " :");

result = myMath.add(firstNumber, secondNumber);

System.*out*.print(result);

**break**;

**case** 2:

System.*out*.println("Substraction of " + firstNumber+" and "

+ secondNumber + " :");

result = myMath.substract(firstNumber, secondNumber);

System.*out*.print(result);

**break**;

**case** 3:

System.*out*.println("Multiple of " + firstNumber + " and "

+ secondNumber + " :");

result = myMath.multiple(firstNumber, secondNumber);

System.*out*.print(result);

**break**;

**case** 4:

System.*out*.println("Division of " + firstNumber + " and "

+ secondNumber + " :");

result = myMath.divide(firstNumber, secondNumber);

System.*out*.print(result);

**break**;

**default**:

System.*out*.println("Invalid operation choice.");

**break**;

}

}

}

**Learning:**

* More usage of scanner; and practice of switch

**STRING TOKENIZER**

**LAB 6.4**

*Rewrite Lab 6.2 – this time user will enter many number at one go (separated by either space or comma or any special character), and output will be sorted list. Do it with the help of StringTokenizer.*

**Learning:**

* Write a SortTheNumbersUsingTokenizer class. Having p.s.v.main method
* Receive numbers to be sorted as user’s input using StringTokenizer
* Save all numbers in an array
* Apply sorting logic and change the numbers in array in sorted order, show result

**package** cg.javaflp.practice;

**import** java.util.Scanner;

**import** java.util.StringTokenizer;

**public** **class** SortTheNumbersUsingTokenizer {

/\*\* Start point of application

\*/

**public** **static** **void** main(String[] args) {

// Using scanner to take input values (numbers) to be sorted

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Please enter the numbers to be sorted, " +

"use comma(,) to separate the number");

String numbers = sc.nextLine();

// Determining count of numbers

**int** length = 0;

StringTokenizer st1 = **new** StringTokenizer(numbers, ",");

**while** (st1.hasMoreTokens()) {

st1.nextToken();

length++;

}

// Creating a array of size equal to count of numbers

Double arr[] = **new** Double[length];

/\* Logic to sort many numbers

\* 1. store the numbers as an array

\* 2. compare each element (number) to its next element (number)

\* 3. replace the numbers if first no is greater then second

\* 4. after step 3, array will be modified

\* 5. iterate step 2 and 3, "length of array" times

\*/

// step 1 : taking numbers as user input and storing in array

**int** arrPosition = 0;

StringTokenizer st = **new** StringTokenizer(numbers, ",");

**while** (st.hasMoreTokens()) {

**if** (arrPosition == length)

**break**;

// System.out.println(st.nextToken());

arr[arrPosition] = Double.*parseDouble*(st.nextToken());

arrPosition++;

}

// This is just for information (JFI) message for user

System.*out*.println("Nos to be sorted : ");

**for**(**int** k = 0; k < arr.length; k++){

System.*out*.print(arr[k] + ", ");

}

// outer FOR loop is for step 5

**for** (**int** i = 0; i < arr.length; i++){

// inner FOR loop is for steps 2, 3 and 4

**for** (**int** j = 1 ; j <arr.length; j++){

Double temp;

**if** (arr[j - 1].compareTo(arr[j])>0){

temp = arr[j - 1];

arr[j-1] = arr[j];

arr[j] = temp;

}

}

}

// Final output (sorted array)

System.*out*.println("\n Sorted value is : ");

**for**(**int** k = 0; k < arr.length; k++){

System.*out*.print(arr[k] + ", ");

}

}

}

**OUTOUT**

Please enter four numbers one by one, use comma(,) to separate the number

12,3,45,23,65,74,17,25,85,123,47,347,2,1346,74,89,12345

Nos to be sorted :

12.0, 3.0, 45.0, 23.0, 65.0, 74.0, 17.0, 25.0, 85.0, 123.0, 47.0, 347.0, 2.0, 1346.0, 74.0, 89.0, 12345.0,

Sorted value is :

2.0, 3.0, 12.0, 17.0, 23.0, 25.0, 45.0, 47.0, 65.0, 74.0, 74.0, 85.0, 89.0, 123.0, 347.0, 1346.0, 12345.0,

**Learning:**

* How and when to use StringTokenizer

**LAB 6.5**

*Modify Lab 6.1; this time – system will validate the data entered by user (Name, age and birth place). Basically now system will validate the data, and will not allow invalid info. (NAME and BIRTH PLACE must be text and AGE must be numbers only).*

*Do this using various methods like charAt, split, etc of string and isLetter, isDigit, etc. of Stringclass.*

*Aim of this LAB is to understand – how to validate various value entered from user and use of various methods of string and other wrapper class*

**Learning:**

* Here, we will write two different class, for better programming practice.
* First, a ValidateFormFields class, which have only static method for validating text and numbers; carefully write this class, as it will be useful for many future assignments.
* Second, a FormValidatePractice class, which will have p.s.v.main method. This is similar to class of Lab assignment 6.1; expect here it will call method of “ValidateFormFields” to validate it

**ValidateFormFields Class**

**package** cg.javaflp.practice;

**public** **class** ValidateFormFields {

/\*\*This method is use to validate a text field input data

\* **@param** text - input value to be validated as text

\* **@return** - true (if data is valid text)

\*/

**public** **static** **boolean** isValidText(String text){

**boolean** response = **true**;

**for** (**int** i = 0; i < text.length(); i++){

**boolean** isValidChar = Character.*isLetter*(text.charAt(i));

**if** (!isValidChar){

response = **false**;

}

}

**return** response;

}

/\*\*This method is use to validate a numeric field input data

\* **@param** text - input value to be validated as number

\* **@return** - true (if data is valid number)

\*/

**public** **static** **boolean** isValidNumber(String text){

**boolean** response = **true**;

**for** (**int** i = 0; i < text.length(); i++){

**boolean** isValidChar = Character.*isDigit*(text.charAt(i));

**if** (!isValidChar){

response = **false**;

}

}

**return** response;

}

}

**Main Class**

**package** cg.javaflp.practice;

**import** java.util.Scanner;

**public** **class** FormValidatePractice {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.*in*);

String name = "";

String birthPlace = "";

**int** age = 0;

/\* We will use do-while loop for all input values as

\* User needs to re-enter the value, until enters valid data

\*/

**boolean** isValidEntry = **false**;

//receiving user's name and validating it

**do** {

System.*out*.println("Please enter your name: ");

name = sc.next();

isValidEntry = ValidateFormFields.*isValidText*(name);

} **while** (!isValidEntry);

// receiving and validating birth place

**do** {

System.*out*.println("Please enter your Birth place: ");

birthPlace = sc.next();

isValidEntry = ValidateFormFields.*isValidText*(birthPlace);

} **while** (!isValidEntry);

// receiving and validating user's age

**do** {

System.*out*.println("Please enter your age: ");

String ageStr = sc.next();

isValidEntry = ValidateFormFields.*isValidNumber*(ageStr);

**if** (isValidEntry){

age = Integer.*parseInt*(ageStr);

}

} **while** (!isValidEntry);

// Counting characters of birth place (city) name

**int** length = birthPlace.length();

age = age + length; // modified the age

// displaying output massage

System.*out*.println("Hi, " + name + " your actual age is "

+ age + "\n" + "MUAHAHAHHAHA");

}

}

**LAB 6.6**

*Create calculator, where user will directly enter numbers and arithmetic operations (of any combination) and system will produce correct result. Here the system should handle any unwanted value entered by user (by validation)*

**DATE**

**LAB 6.7**

*Create 2 date objects using two constructors of date (use one for each object), and run its methods (after, before, compare, getTime, setTime).*

**package** cg.javaflp.practice;

**import** java.util.Date;

**public** **class** DatePractice {

/\*\*

\* Main method - starting point of application

\*/

**public** **static** **void** main(String[] args) {

// creating date object by its two constructors

Date date1 = **new** Date();

System.*out*.println("Date1 : " + date1);

Date date2 = **new** Date(34567892L);

System.*out*.println("Date2 : " + date2);

// use of methods of Date class

**int** resultCompare = date2.compareTo(date1);

System.*out*.println(resultCompare);

**boolean** result = date2.after(date1);

System.*out*.println(result);

result = date2.before(date1);

System.*out*.println(result);

System.*out*.println(date1.getTime());

date2.setTime(1434568923457L);

System.*out*.println(date2);

}

}

**Learning**

* Two different constructor of Date class
* Various method of Date class

**LAB 6.8**

Create a date object, print it on console; now create a SDF object, and format the date object, print it. Check the display of date is changed as format. Do the same for few more formats.

**Learning**

* Write a DatePractice class
* Create a date object
* Create a simple date format for a pattern
* Change format of date object
* Repeat above two steps for various formats

**package** cg.javaflp.practice;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** DatePractice {

/\*\* Main method - starting point of application

\*/

**public** **static** **void** main(String[] args) {

// creating date object by its two constructors

Date date1 = **new** Date();

System.*out*.println("Date1 : " + date1);

/\* Formatting of date with the help of simple date format

\*/

String pattern = "dd/MM/yyyy";

SimpleDateFormat sdf = **new** SimpleDateFormat(pattern);

System.*out*.println("Date1: " + sdf.format(date1));

String pattern1 = "MM/dd/yyyy";

SimpleDateFormat sdf1 = **new** SimpleDateFormat(pattern1);

System.*out*.println("Date1: " + sdf1.format(date1));

}

}

**Learning**

* How and when to use SDF
* How to apply and use various format

**LAB 6.9**

*Create a SDF object, and parse and String (of given format) into date object.*

**Learning**

* Modify DatePractice class
* Create a string object, assign it a valid date text
* Create simple date format (sdf)for a given pattern
* Parse the string into date for the sdf
* Repeat above two steps for more patterns

**package** cg.javaflp.practice;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** DatePractice {

/\*\* Main method - starting point of application

\*/

**public** **static** **void** main(String[] args) {

String date3Str = "03/28/2006";

String pattern1 = "MM/dd/yyyy";

SimpleDateFormat format = **new** SimpleDateFormat(pattern1);

**try** {

Date date3 = format.parse(date3Str);

System.*out*.println("Date from string parsing : " + date3);

} **catch** (ParseException e) {

System.*out*.println("Could not parse………");

e.printStackTrace();

}

}

}

Add code below inside try catch block of parsing logic – here, we are checking the parsing of same string value of different simple date format object -

Date date4 = sdf.parse(date3Str);

System.*out*.println("same Date from diff sdf parsing : " + date4);

**Learning**

* How to Parse String into date

**LAB 6.10**

*Practice Java UTIL date and Java SQL date – create both UTIL date and SQL date and print/display it on console, check what is difference between them. Create a SQL date from UTIL date and vice versa.*

// create a SQL date

java.sql.Date sqlDate = **new** java.sql.Date(1231231244L);

System.*out*.println(sqlDate);

// Create an UTIL Date

Date today = **new** Date();

System.*out*.println(today);

// create an UTIL date from SQL date

Date utilDate = java.sql.Date.*valueOf*(sqlDate.toString());

System.*out*.println(utilDate);

// convert an UTIL date into SQL date (it cannot be class casted)

sqlDate = **new** java.sql.Date(today.getTime());

System.*out*.println(sqlDate);

**Learning**

* Difference between SQL date and UTIL date
* How to create one from another; class-casting will not work here

**CALENDAR**

**LAB 6.11**

*Create a calendar object using Calendar.getInstance(), and practice/execute at least 15 methods of calendar class (Ref to calendar API on oracle website* [*http://download.oracle.com/javase/1.4.2/docs/api/java/util/Calendar.html*](http://download.oracle.com/javase/1.4.2/docs/api/java/util/Calendar.html)*)*

**package** cg.javaflp.practice;

**import** java.util.Calendar;

**public** **class** CalendarPractice {

**public** **static** **void** main(String[] args) {

Calendar cal = Calendar.*getInstance*();

// setting date of calendar and printing its date

cal.set(2009, 1, 13);

System.*out*.println(cal); // print obj ref

System.*out*.println(cal.getTime()); // print date

cal.set(Calendar.*YEAR*, 2012); // changing its year

System.*out*.println(cal.getTime());

// similar for date and month

cal.set(Calendar.*MONTH*, 8);

cal.set(Calendar.*DAY\_OF\_MONTH*, 9);

System.*out*.println(cal.getTime());

// how to get values

System.*out*.println(cal.get(Calendar.*DATE*));

System.*out*.println(cal.get(Calendar.*DAY\_OF\_MONTH*));

System.*out*.println(cal.get(Calendar.*DAY\_OF\_WEEK*));

System.*out*.println(cal.get(Calendar.*DAY\_OF\_YEAR*));

System.*out*.println(cal.get(Calendar.*DAY\_OF\_WEEK\_IN\_MONTH*));

System.*out*.println(cal.get(Calendar.*WEEK\_OF\_MONTH*));

System.*out*.println(cal.get(Calendar.*WEEK\_OF\_YEAR*));

// changing time of the day

cal.set(Calendar.*HOUR\_OF\_DAY*, 16);

System.*out*.println(cal.getTime());

}

}

**Learning**

* How to get calendar’s instance and its various methods
* How to use it (Set values and get values)

**LAB 6.12**

*Create a Gregorian calendar object and practice/execute at least 15 methods of calendar class (Ref to Gregorian calendar API on oracle website)*

**package** cg.javaflp.practice;

**import** java.util.Calendar;

**import** java.util.GregorianCalendar;

**import** java.util.Locale;

**import** java.util.TimeZone;

**public** **class** GregorianCalendarPractice {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

GregorianCalendar cal = **new** GregorianCalendar();

System.*out*.println(cal.getTime());

cal.set(GregorianCalendar.*DAY\_OF\_YEAR*, 246);

System.*out*.println(cal.getTime());

cal.set(GregorianCalendar.*DAY\_OF\_WEEK*, GregorianCalendar.*TUESDAY*);

System.*out*.println(cal.getTime());

/\*

GregorianCalendar cal2 = new

GregorianCalendar(TimeZone.getTimeZone("America/Chicago"), Locale.US);

System.out.println(cal2.getTime());

GregorianCalendar calUK = new GregorianCalendar(Locale.UK);

System.out.println(calUK.getTime());

\*/

// use of roll to change current date

cal.roll(Calendar.*MONTH*, **false**);

System.*out*.println(cal.getTime());

cal.roll(Calendar.*MONTH*, -4);

System.*out*.println(cal.getTime());

// comparing two calendar objects

GregorianCalendar today = **new** GregorianCalendar();

System.*out*.println(today.after(cal));

System.*out*.println(today.before(cal));

System.*out*.println(today.compareTo(cal));

}

}

**LAB 6.13**

*Print the days of weeks, using “DateFormatSymbols” class.*

String[] weekdays = **new** DateFormatSymbols().getWeekdays();

**for**(String s: weekdays){

System.*out*.println(s);

}

**LAB 6.14**

*Write a program where system will ask for user’s date of birth (take input with the help of scanner) and then as a response system will tell – user was born of which day and in current year user’s birthday will fall on which day (user Gregorian calendar for this).*

**package** cg.javaflp.practice;

**import** java.text.DateFormatSymbols;

**import** java.util.GregorianCalendar;

**import** java.util.Scanner;

/\*\*

\* This class is use to tell a users age, on which day user were born

\* And this, year his/her birthday will be on which day

\*/

**public** **class** DayPredictor {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

// Receiving user input for user's data of birth

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Plz enter your age in dd/MM/yyyy format :");

String birthdateStr = sc.next();

String[] strArr = birthdateStr.split("/");

**int** year = Integer.*parseInt*(strArr[2]);

**int** month = Integer.*parseInt*(strArr[1]);

**int** dayOfMonth = Integer.*parseInt*(strArr[0]);

// Creating calendar (date) object with the help of user data

GregorianCalendar cal = **new** GregorianCalendar(year, month, dayOfMonth);

// Finding out day of user's birth date

String[] weekdays = **new** DateFormatSymbols().getWeekdays();

String birthDay = weekdays[cal.get(GregorianCalendar.*DAY\_OF\_WEEK*)];

// Finding age of user as of today

GregorianCalendar today = **new** GregorianCalendar();

**int** age = today.get(GregorianCalendar.*YEAR*)

- cal.get(GregorianCalendar.*YEAR*);

// Finding out day of user's birth date in current year

cal.set(GregorianCalendar.*YEAR*, 2012);

String thisYearDay = weekdays[cal.get(GregorianCalendar.*DAY\_OF\_WEEK*)];

System.*out*.println("You were born on " + birthDay);

System.*out*.println("Today, you are " + age + " years old");

System.*out*.println("This year, your birthday will be on "+thisYearDay);

}

}

**OUTPUT**

Plz enter your age in dd/MM/yyyy format :

14/10/1980

You were born on Friday

Today, you are 32 years old

This year, your birthday will be on Wednesday

**REG EX (Regular Expression)**

**LAB 6.15**

*Rewrite Lab 6.2 – this time use RegEx to validate the data entered by user*

**Steps:**

* Write a method in isValidNumberUsingRegEx(String number) in ValidateFormFields class of Lab assignment 6.5
* Modify the SortTheNumber class or class of Lab assignment 6.3; Now after receiving every number system will check if it is a valid number
* To check it, this class will use “isValidNumberUsingRegEx” method of ValidateFormFields class

**Method in ValidateFormFields Class**

/\*\*

\* This method is use to validate a numeric input data using RegEx

\* **@param** text - input value to be validated as number

\* **@return** - true (if data is valid number)

\*/

**public** **static** **boolean** isValidNumberUsingRegEx(String text){

**return** text.matches("[0-9]\*");

}

**public** **static** **boolean** isValidNameUsingRegEx(String text){

**return** text.matches("[A-Z a-z]\*");

}

**Steps:**

* How to use RegEx to validate numeric data

**ANNOTATION**

**LAB 6.16**

*Create an annotation type element; use it as class level comment for a class.*

**LAB 6.17**

*Convert the annotation type element of Lab X.1 to appear in Javadoc-generated documentation*

**LAB 6.18**

*Use @Deprecated annotation on any method of a class; now call this method from any other class.*

*See what happen (system will show warning message – that method is deprecated)*

**LAB 6.19**

*Create subclass of any class – use @override annotation on any method which subclass is overriding. Try to change the method signature (change parameter load).*

*See what happen (system will not allow to overload the method, as @override annotation is there, so it can only be overrided)*

**LAB 6.19**

*In Lab X.3 use @SuppressWarnings("deprecation") annotation in the method of calling class (which is calling the @deprecated method of Lab X.1).*

*See what happen (system will remove the deprecated warning message)*

**LAB 6.20**

*Write a program, where user will fill a registration form (with all basic info – Name, password, confirm password, data of birth, phone number and email id); Validate the form and save the info in a user object (POJO) and then store this object in a user-stack (array).*

*System will check the form at the time user will ask to save it. System should show appropriate message in case of invalid data. Upon successful submission, system will display the data of the entire user entered so far in user stack (array).*

**LAB 6.21**

*Modify the lab 6.X2.*

*Now, system should check the data entered by the user, at that field only; and if it is invalid, it should not allow user to enter the data for other field. In other word, instead of at the time of submitting the form, now data is being validating at the time of entry only.*