**LAB 5.1**

*Check the java math API on oracle’s (previously SUN) website (*<http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html>*), and use at least 15 method of Math class in your main class.*

**Steps:**

* Write a class MathPrac; write p.s.v. main method
* Go to <http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html>
* Create few int, float, double variables and use various elementary exponential, logarithm, square root, and trigonometric functions of Math class

/\*\*

\* This class is use to practice various methods of

\* MATH class

\*/

**public** **class** MathPractice {

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

System.*out*.println(Math.*abs*(44.5));

System.*out*.println(Math.*E*);

System.*out*.println(Math.*PI*);

System.*out*.println(Math.*pow*(3, 4));

System.*out*.println(Math.*sqrt*(55));

System.*out*.println(Math.*floor*(-55.8));

System.*out*.println(Math.*nextUp*(55));

System.*out*.println(Math.*tan*(45));

System.*out*.println(Math.*toDegrees*(Math.*tan*(45)));

}

}

**Learning:**

* All method of method class is Static
* Math class have two properties E and Pi
* Understand how to use various elementary exponential, logarithm, square root, and trigonometric functions of Math class

**LAB 5.2**

*Check the java math API on oracle’s (previously SUN) website), and use at least 15 method of String class in your main class.*

**N/A**

**Just practice various method of String and create string object using its various constructors**

**Same as we did for Lab 5.1**

**LAB 5.3**

Instantiate a String object by two ways

1. calling new operator (String s1 = new String(“flp”))
2. by directly assigning a String value (String s2 = “flp”)

Check whether s1 and s2 are same object or different. (use, == and equals() method )

/\*\*

\* This class is use to practice various method and behavior of string class

\*/

**public** **class** StringPractice {

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

StringPractice practice = **new** StringPractice();

practice.stringBerhavior();

}

**private** **void** stringBerhavior(){

String s1 = "abc";

String s2 = **new** String("abc");

// check if s1 and s2 are same object (pointing to same ref)

System.*out*.println(s1 == s2);

// check if object value of s1 and s2 is same

System.*out*.println(s1.equals(s2));

}

}

**Learning:**

* Direct assigning a text value to string is pointing to string pool
* String has overridden equal method, to compare object equality on basis of its value

**LAB 5.4**

*Write a Stack class, using array; it should have Push and Pop method, and size of array will be assigned by calling class (i.e. each object of this stack class can have different stack size). Use this stack class to push and pop data*

*Now write a class (having main method) which will use this stack to push some values; now this class also show the data pushed into stack, The output info should be in a single line (user string buffer to append pop values and show the result)*

**Steps:**

* Write a class StackPrac; which will work on LIFO theory. It will have four methods
  + pop - it will pop the top object in stack, if available; return type Object
  + push - it will push an object into stack, if space will be available in stack; return void
  + isEmpty –to check if Stack is empty (use in pop method); return type boolean
  + isFull –to check if Stack is full (use in push method); return type boolean
* This stackPrac class should be a perfectly written class, as we will use to a lot in future assignments.
* Now, in MainClass (class having main method) create an object of StackPrac and push some data; then pop some data; Use StringBuilder or StringBuffer to show output in a single line.

**STACK PRAC CLASS**

**package** cg.javaflp.practice;

/\*\*

\* This class is basically creating a stack object

\* having push and pop behaviour and works on pricipal of LIFO

\*

\* Instantiator of this object will define the size of stack

\*/

**public** **class** StackPrac {

**int** top;

Object [] objArr;

/\*\*

\* Constructor of stack; it creates an empty stack and set it size

\* **@param** size - to define size of stack

\*/

**public** StackPrac(**int** size) {

objArr = **new** Object[size];

top = -1;

}

// to check if stack is empty

**private** **boolean** isEmpty(){

**return** top < 0;

}

// to check if stack is full

**private** **boolean** isFull(){

**return** top >= objArr.length;

}

/\*\*

\* method to Push objects in stack

\* it will not allow push, if stack is full

\* **@return** boolean value for confirmation

\*/

**public** **boolean** push(Object obj){

**if** (isFull()){

**return** **false**;

}

top ++;

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

objArr[top] = obj;

**return** **true**;

}

/\*\*

\* method to Pop objects from stack

\* it will not allow pop, if stack is empty

\* **@return** popped out Object,

\* in case of empty stack it will return null

\*/

**public** Object pop(){

**if** (isEmpty()){

**return** **null**;

}

Object poppedObj = objArr[top];

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

top -- ;

**return** poppedObj;

}

}

**Class that uses StackPrac (stack object)**

**package** cg.javaflp.practice;

/\*\*

\* This class uses stackPrac (stack obejct) to push and pop data

\* it is also used to practice stringBuilder (stringBuffer)

\*/

**public** **class** UseStack {

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

// create a stack of size 5

StackPrac stack = **new** StackPrac(5);

// pushing three string objects

stack.push("First element");

stack.push("another");

stack.push("third element");

StringBuilder sb = **new** StringBuilder();

//popping out objects value and then appending to string builder

sb.append(stack.pop());

sb.append(" and ");

sb.append(stack.pop());

sb.append(" and now the ");

sb.append(stack.pop());

System.*out*.println(sb);

}

}

**Learning:**

* How to create a software stack
* Use of StringBuilder (StringBuffer)

**LAB 5.5**

*Write a program, which takes multiple numeric values from command line and program, will sort them into (ascending) acsc and desc (descending) order and display*

**Steps:**

* Pass various numbers as runtime argument
* Parse them into appropriate data type
* Use compare method, to compare the numbers and sort them

**package** cg.javaflp.practice;

/\*\*

\* This class is use to sort the numbers in

\* ascending and descending order

\*

\* Numbers will be pass on run time as command line args

\*/

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

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

// receiving command line values and parsing it as number

Double a = Double.*parseDouble*(args[0]);

Double b = Double.*parseDouble*(args[1]);

Double c = Double.*parseDouble*(args[2]);

Double d = Double.*parseDouble*(args[3]);

// displaying that users have passed these numbers

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 than 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**

Four numbers to be sorted..

88.0, 23.0, 56.0, 4.0

Sorted value is :

4.0, 23.0, 56.0, 88.0

**LAB 5.6 (8 assignment within this)**

*Check the java API for all wrapper classes on oracle’s (previously SUN) website, and use at least 15 method of all wrapper classes separately.*

**N/A**

**Do the similar exercise as we did for Lab 5.1 – this time it will have 8 such practice classes for following**

Integer

Char

Short

Byte

Float

Double

Boolean

Long

**Learning:**

* Understanding of various method of Wrapper classes
* Use of important methods like - compare, toString, parses, etc.

**LAB 5.7**

*Create two variables of all Wrapper classes and initialize them with some value (Integer, Float, Long, Double, Boolean, Short, Byte and Character). Now use the comparator method of each Wrapper class to compare the respective variables, and come up with matrix of output in all case.*

**public** **class** WrapperPractice {

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

// creating two Double object, and checking compareTo behavior

Double firstDouble = 44.9;

Double anotherDouble = 34.98;

System.*out*.println("Double"+firstDouble.compareTo(anotherDouble));

// creating two Float object, and checking compareTo behavior

Float firstFloat = 34.1F;

Float anotherFloat = 56.7F;

System.*out*.println("Float "+firstFloat.compareTo(anotherFloat));

// creating two Integer object, and checking compareTo behavior

Integer firstInterger = 21;

Integer anotherInteger = 12;

System.*out*.println("Interger"+firstInterger.compareTo(anotherInteger));

// creating two Byte object, and checking compareTo behavior

Byte firstByte = 45;

Byte anotherByte = 23;

System.*out*.println("Byte " + firstByte.compareTo(anotherByte));

// creating two Short object, and checking compareTo behavior

Short firstShort = 67;

Short anotherShort = 88;

System.*out*.println("Short "+ firstShort.compareTo(anotherShort));

// creating two Long object, and checking compareTo behavior

Long firstLong = 4568L;

Long anotherLong = 76813L;

System.*out*.println("Long " + firstLong.compareTo(anotherLong));

// creating two Character object, and checking compareTo behavior

Character c1 = 'a';

Character c2 = 'd';

System.*out*.println("Char " + c1.compareTo(c2));

// creating two Boolean object, and checking compareTo behavior

Boolean b1 = **true**;

Boolean b2 = **false**;

System.*out*.println("Boolean " + b1.compareTo(b2));

}

}

**OUTPUT**

Double 1

Float -1

Interger 1

Byte 22

Short -21

Long -1

Char -3

Boolean 1

**Learning:**

* Understand comparator method of Wrapper classes
* Output of comparator method of different Wrapper class
* Matrix

|  |  |
| --- | --- |
| **Wrapper class** | **Output of comparator method** |
| Character | Difference of value |
| Integer | 1 if first value > second  0 if first = second  -1 if first < second |
| Short | Difference of value (first value – second) |
| Byte | Difference of value (first value – second) |
| Long | 1 if first value > second  0 if first = second  -1 if first < second |
| Float | 1 if first value > second  0 if first = second  -1 if first < second |
| Double | 1 if first value > second  0 if first = second  -1 if first < second |
| Boolean | Difference of value (first value – second) |