|  |
| --- |
| **Object Oriented Programming using JAVA Lab** |

**Subject Code :18CSI302L Credits : 02 L-T-P: 0-0-4**

**List of Experiments:**

1. Demonstrate Constructor Overloading and Method Overloading in JAVA.

1. Implement Inner Classes and demonstrate its access protection.
2. Implement the following:
   1. An abstract class "Shape" with the following properties: an instance variable shapeName of type String, an abstract method area (), a toString() method that returns the name of the shape.
   2. Create a subclass named "Sphere" which has radius and its area given by the formula 4\*PI\*r^2.
   3. Create a subclass named "Rectangle" which has length and width and its area is length times width.
   4. Create a subclass named "Triangle" which has base and height and its area is ½\*base\*height.
   5. Create another class which displays the calculated area.
3. Perform the following operations:
   1. Check the length and capacity of String and StringBuffer objects
   2. Reverse the contents of a string given on console and convert the resultant string in Upper Case.
   3. Input a string from the console and append it to above resultant string.
   4. Extract the substring from resultant string.
4. Create the following:
   1. A class "Account" with minimum balance 1000rs, deposit() method to deposit amount, withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance less than 1000rs.
   2. A class "LessBalanceException" which returns the statement that says "withdraw amount ( \_\_ rs) is not valid".
   3. A class which creates 2 accounts through which both deposit and withdraw operations are performed. Appropriate action has to be taken for LessBalanceException.
5. Implement Linear Queue using user defined exception handling (also use 'throw' and ‘throws’ keyword)
6. Implement the concept of Producer Consumer using synchronized threads.
7. Create the following:
   1. Create an Interface for ‘Stack’ operations.
   2. A class that implements the Stack interface and create a fixed length stack.
   3. A class that implements the Stack interface and create a dynamic length stack.
   4. A class that uses the above stacks through interface reference and does the stack operations that demonstrates the runtime binding
8. Develop the following:
   1. Create a package named "Calculator".
   2. Create some classes in the package representing some common operations like addition, subtraction, multiplication and division.
   3. Import and compile these classes in other program.
9. Create an enumeration DayofWeek with seven values SUNDAY through SATURDAY. Add a method isWorkday() to the DayofWeek class that returns true if the value on which it is called is MONDAY through FRIDAY.
10. Using File I/O streams, write a program to demonstrate file operations.
11. Write a Swing Application which uses:
    1. JTabbed Pane
    2. Each Tab should use JPanel, which includes any one component given
       1. below in each Panel
       2. ComboBox / List / Tree / Radiobutton

**1.** Demonstrate Constructor Overloading and Method Overloading in JAVA.

**//Method Overloading**

**class** Overload  
{  
 **int a**=0,**b**=0;  
  
 **void** add()  
 {  
 System.***out***.println(**"NO or DEFAULT argument add method, sum= "**+(**a**+**b**));  
 }  
 *//overloaded Method add* **void** add(**int** k,**int** j)  
 {  
 **a**=k;  
 **b**=j;  
 System.***out***.println(**"Parameterized argument add method, sum= "**+(**a**+**b**));  
 }  
  
}  
  
**public class** One {  
 **public static void** main(String args[])  
 {  
 Overload ga=**new** Overload();  
 System.***out***.println(**"Default Method call"**);  
 ga.add();  
 System.***out***.println(**"Parameterized Method call"**);  
 ga.add(12,23);  
 }  
}

**OUTPUT**

Default Method call

NO or DEFAULT argument add method, sum= 0

Parameterized Method call

Parameterized argument add method, sum= 35

**//Constructor Overloading**

**class** Generic  
{  
 **int a**=0,**b**=0;  
  
 Generic()  
 {  
 **a**=**b**=1;  
 }  
 *//overloaded constructor* Generic(**int** a,**int** b)  
 {  
 **this**.**a**=a;  
 **this**.**b**=b;  
 }  
  
 **void** print()  
 {  
 System.***out***.println(**"a="**+**a**+**" b="**+**b**);  
 }  
  
}  
  
**public class** One {  
 **public static void** main(String args[])  
 {  
 System.***out***.println(**"Default constructor call"**);  
 Generic ga=**new** Generic();  
 ga.print();  
 System.***out***.println(**"Parameterized constructor call"**);  
 Generic ag=**new** Generic(10,20);  
 ag.print();  
 }  
}

**OUTPUT**

Default constructor call

a=1 b=1

Parameterized constructor call

a=10 b=20

2. Implement Inner Classes and demonstrate its access protection.

**import** java.io.\*;  
  
**class** Outer  
{  
 **int outdata** = 22;  
  
 **void** display()  
 {  
 Inner inobj = **new** Inner();  
 System.***out***.println(**"Accessing from outer class"**);  
 System.***out***.println(**"The value of outdata is "** +**outdata**);  
 System.***out***.println(**"The value of indata is "** +inobj.**indata**);  
 }  
  
 **class** Inner  
 {  
 **int indata** = 20;  
 **void** inmethod()  
 {  
 System.***out***.println(**"Accessing from inner class"**);  
 System.***out***.println(**"The sum of indata & outdata is "** +(**outdata** + **indata**));  
 }  
 }  
}  
  
**public class** Two {  
 **public static void** main(String args[])  
 {  
 Outer outobj = **new** Outer();  
 outobj.display();  
 Outer.Inner inobj1 = outobj.**new** Inner();  
 inobj1.inmethod();  
 }  
}

**OUTPUT**

Accessing from outer class

The value of outdata is 22

The value of indata is 20

Accessing from inner class

The sum of indata & outdata is 42

**3.** Implement the following:

* 1. An abstract class "Shape" with the following properties: an instance variable shapeName of type String, an abstract method area (), a toString() method that returns the name of the shape.
  2. Create a subclass named "Sphere" which has radius and its area given by the formula 4\*PI\*r^2.
  3. Create a subclass named "Rectangle" which has length and width and its area is length times width.
  4. Create a subclass named "Triangle" which has base and height and its area is ½\*base\*height.
  5. Create another class which displays the calculated area.

**import** java.util.Scanner;  
**abstract class** Shape  
{  
 **final double PI**=3.14; *//constant* String **shapeName**;  
 **abstract void** area(); *//abstract method* **public** String toString() *//tostring method to return calling classname* {  
 **return this**.getClass().getName();  
 }  
}  
  
**class** Sphere **extends** Shape  
{  
 **double rad**;  
 Sphere(**double** r) *//constructor to set radius* {  
 **rad**=r;  
 }  
  
 **void** area()  
 {  
 System.***out***.println(**"Area = "**+(4\***PI**\***rad**\***rad**)); *//area of sphere formula* }  
}  
  
**class** Rectangle **extends** Shape  
{  
 **double len**, **width**;  
  
 Rectangle(**double** l,**double** w)  
 {  
 **len**=l;  
 **width**=w;  
 }

**void** area()  
 {  
 System.***out***.println(**"Area = "**+(**len**\***width**));  
 }  
}  
  
**class** Triangle **extends** Shape  
{  
 **double base**, **height**;  
  
 Triangle(**double** b,**double** h)  
 {  
 **base**=b;  
 **height**=h;  
 }  
  
 **void** area()  
 {  
 System.***out***.println(**"Area = "**+(0.5\***base**\***height**));  
 }  
  
}  
  
**public class** Three {  
 **public static void** main(String args[])  
 {  
 Scanner s=**new** Scanner(System.***in***);  
 System.***out***.println(**"Enter the radius of a circle"**);  
 **double** r=s.nextInt(); *//taking radius input* Sphere sp=**new** Sphere(r); *//sphere object creation* System.***out***.println(sp.toString()); *//respective class object calls tostring()* sp.area(); *//calling respective class area method* System.***out***.println(**"Enter the len and wid: "**);  
 **double** l=s.nextInt();  
 **double** w=s.nextInt();  
 Rectangle rc=**new** Rectangle(l,w);  
 System.***out***.println(rc.toString());  
 rc.area();  
  
 System.***out***.println(**"Enter the base and height: "**);  
 **double** b=s.nextInt();  
 **double** h=s.nextInt();  
 Triangle ta=**new** Triangle(b,h);  
 System.***out***.println(ta.toString());  
 ta.area();  
 }  
}

**OUTPUT:**

Enter the radius of a circle

2

Sphere

Area = 50.24

Enter the len and wid:

2 2

Rectangle

Area = 4.0

Enter the base and height:

3 2

Triangle

Area = 3.0

1. Perform the following operations:
   1. Check the length and capacity of String and StringBuffer objects
   2. Reverse the contents of a string given on console and convert the resultant string in Upper Case.
   3. Input a string from the console and append it to above resultant string.
   4. Extract the substring from resultant string.

**import** java.util.\*;

**class** Cases  
{  
 *//length and capacity of string and stringbuffer* String **s**=**"Jain University"**; *//String initialization* StringBuffer **sb1**=**new** StringBuffer(); *//String buffer with default size but no input* StringBuffer **sb2**=**new** StringBuffer(**"SETJU"**); *//String buffer with input* StringBuffer **sb3**=**new** StringBuffer(50); *//String buffer with capacity input* **void** demo1()  
 {  
 System.***out***.println(**"Case1"**);  
 System.***out***.println(**"with input, string length is: "**+**s**.length()); *//String length* System.***out***.println(**"Before input, stringbuffer length is: "**+**sb1**.length()); *//StringBuffer length* System.***out***.println(**"With input, stringbuffer length is: "**+**sb2**.length());  
 System.***out***.println(**"With capacity input, stringbuffer length is: "**+**sb3**.length());  
 System.***out***.println(**"Before input, stringbuffer length is: "**+**sb1**.capacity()); */\* StringBuffer*

*capacity \*/* System.***out***.println(**"With input, stringbuffer length is: "**+**sb2**.capacity());  
 System.***out***.println(**"With capacity input, stringbuffer length is: "**+**sb3**.capacity());  
 }  
  
 String demo2(String read)  
 {  
 String reverse = **""**;  
 **int** i=read.length()-1;  
 **while**(i>=0) {  
 reverse = reverse + read.charAt(i--);  
 }  
 **return** reverse;  
  
 }  
  
 String demo3(String read1, String read2)  
 {  
 **return** read1.concat(read2);  
 }

**void** demo4(String read)  
 {  
 System.***out***.println(**"\nCase4"**);  
 System.***out***.println(**"Given only begining index as 2 "**+read.substring(2));  
 System.***out***.println(**"Given begining and end index as (2 to 4) "**+read.substring(2,4));  
 }  
}  
  
**public class** Four {  
 **public static void** main(String args[])  
 {  
 Scanner sc=**new** Scanner(System.***in***);  
  
 Cases c1=**new** Cases();  
 c1.demo1();

System.***out***.println(**"\nCase2"**);  
 System.***out***.println(**"Enter string to reverse: "**);  
 String s1=sc.nextLine();  
 String res1=c1.demo2(s1);  
 System.***out***.println(**"Reversed string is "**+res1);  
  
 System.***out***.println(**"\nCase3"**);  
 System.***out***.println(**"Enter string to append: "**);  
 String s2=sc.nextLine();  
 String res2=c1.demo3(res1,s2);  
 System.***out***.println(**"Concatenated string is "**+res2);  
  
 c1.demo4(res2);  
 }  
}

**OUTPUT**

Case1

with input, string length is: 15

Before input, stringbuffer length is: 0

With input, stringbuffer length is: 5

With capacity input, stringbuffer length is: 0

Before input, stringbuffer length is: 16

With input, stringbuffer length is: 21

With capacity input, stringbuffer length is: 50

Case2

Enter string to reverse:

xyz

Reversed string is zyx

Case3

Enter string to append:

teacher

Concatenated string is zyxteacher

Case4

Given only begining index as 2: xteacher

Given begining and end index as (2 to 4): xt

**5.** Create the following:

* 1. A class "Account" with minimum balance 1000rs, deposit() method to deposit amount, withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance less than 1000rs.
  2. A class "LessBalanceException" which returns the statement that says "withdraw amount ( \_\_ rs) is not valid".
  3. A class which creates 2 accounts through which both deposit and withdraw operations are performed. Appropriate action has to be taken for LessBalanceException.

**import** java.util.\*;  
  
**class** LessBalanceException **extends** Exception //Defining user-defined exception class “LessBalanceException”

{  
 LessBalanceException(String s) //Constructor which takes user error message  
 {  
 **super**(s); //calls super class constructor  
 }  
}  
  
**class** Account //class account  
{  
 **double minbal**;  
 Account()  
 {  
 **minbal**=1000; //minimum balance of any account is made 1000rs  
 }  
  
 **void** deposit(**double** amt)  
 {  
 **minbal**=**minbal**+amt; //deposit amount is added to minimum balance  
 }  
  
 **void** withdraw(**double** amt)  
 {  
 **double** t;  
 t=**minbal**;  
 t=t-amt;  
 **try** {  
 **if** (t < 1000) //check against balance not to be lesser than 1000rs at any time  
 **throw new** LessBalanceException(**"Balance can't go less then 1000 (Withdraw amount is**

**not valid)"**);  
 **else** {  
 **minbal** = **minbal** - amt; //withdraw amount is subtracted from latest minimum balance  
 System.***out***.print(**"Balance after withdraw:"**);  
 System.***out***.println(**minbal**);  
 }  
 }  
 **catch**(LessBalanceException l)  
 {  
 System.***out***.println(l.getMessage());  
 }  
 }  
  
 **double** balance()  
 {  
 **return minbal**;  
 }  
}  
  
**public class** Five {  
 **public static void** main(String args[])  
 {  
 Account a1=**new** Account(); //creating account1  
 Account a2=**new** Account(); //creating account2  
 **double** dep,wit;  
 Scanner sc=**new** Scanner(System.***in***);  
 System.***out***.print(**"Min bal of account1: "**);  
 System.***out***.println(a1.balance()); //account1 minimum balance  
 System.***out***.println(**"Enter the amount to deposit: "**);  
 dep = sc.nextDouble();  
 a1.deposit(dep); //account1 deposit method call  
 System.***out***.print(**"Balance after deposit:"**);  
 System.***out***.println(a1.balance());  
 System.***out***.println(**"Enter the amount to Withdraw: "**);  
 wit = sc.nextDouble();  
 a1.withdraw(wit); //account1 withdraw method call  
  
 System.***out***.print(**"\nMin bal of account2: "**);  
 System.***out***.println(a2.balance()); //account2 minimum balance  
 System.***out***.println(**"Enter the amount to deposit: "**);  
 dep = sc.nextDouble();  
 a2.deposit(dep); //account2 deposit method call  
 System.***out***.print(**"Balance after deposit:"**);  
 System.***out***.println(a2.balance());  
 System.***out***.println(**"Enter the amount to Withdraw: "**);  
 wit = sc.nextDouble();  
 a2.withdraw(wit); //account2 withdraw method call  
 }  
}

**OUTPUT:**

Min bal of account1: 1000.0

Enter the amount to deposit:

2000

Balance after deposit:3000.0

Enter the amount to Withdraw:

2500

Balance can't go less then 1000 (Withdraw amount is not valid)

Min bal of account2: 1000.0

Enter the amount to deposit:

3000

Balance after deposit:4000.0

Enter the amount to Withdraw:

2000

Balance after withdraw:2000.0

6. Implement Linear Queue using user defined exception handling (also use 'throw' and ‘throws’ keyword)

**class** LinearQueue **extends** Exception //user-defined exception class “LinearQueue”  
{  
 **int front**,**rear**;  
 **final int size**=5;  
 **int q**[]=**new int**[**size**];

LinearQueue()  
 {  
 **front**=0;  
 **rear**=-1;  
 }

LinearQueue(String s)  
 {  
 **super**(s); //calling constructor of super class Exception  
 }  
  
 **void** enqueue(**int** ele) **throws** LinearQueue  
 {  
 **try** {  
 **if** (**rear** == -1 || **rear** < **size**-1) {  
 **q**[++**rear**] = ele;  
 System.***out***.println(**"Q has "**+**q**[**rear**]+ **" at pos "**+**rear**);  
 }  
 **else** {  
 **throw new** LinearQueue(**"Queue is full\n"**);  
 }  
 }  
 **catch**(LinearQueue lq) {  
 System.***out***.println(lq.getMessage());  
 }  
 }  
  
 **void** dequeue() **throws** LinearQueue  
 {  
 **try** {  
 **if** (**front** < **size**) {  
 System.***out***.println(**"Deleted element is "** + **q**[**front**++]);  
 }  
 **else** {  
 **if**(**front**==**size**)  
 **front**=-1;  
 **throw new** LinearQueue(**"No elements to delete"**);  
 }  
 }  
 **catch**(LinearQueue lq)  
 {  
 System.***out***.println(lq.getMessage());  
 }  
 }  
  
 **void** display()  
 {  
 **if**(**front**>-1)  
 {  
 System.***out***.println(**"Elements are"**);  
 **int** k=**front**;  
 **for**(**int** i=k;i<=**rear**;i++)  
 System.***out***.println(**q**[k++]);  
 }  
 }  
}  
  
**public class** Six

{  
 **public static void** main(String args[]) **throws** LinearQueue  
 {  
 LinearQueue q=**new** LinearQueue();  
 q.enqueue(10);  
 q.enqueue(20);  
 q.enqueue(30);  
 q.enqueue(40);  
 q.enqueue(50);  
 q.display();  
 System.***out***.println(**"When tried to put sixth element to full queue"**);  
 q.enqueue(60);  
  
 q.dequeue();  
 q.dequeue();  
 q.dequeue();  
 q.display();  
 q.dequeue();  
 q.dequeue();  
 System.***out***.println(**"When tried to remove sixth element from empty queue"**);  
 q.dequeue();  
 q.display();  
 }  
}

**OUTPUT**

Q has 10 at pos 0

Q has 20 at pos 1

Q has 30 at pos 2

Q has 40 at pos 3

Q has 50 at pos 4

Elements are

10

20

30

40

50

When tried to put sixth element to full queue

Queue is full

Deleted element is 10

Deleted element is 20

Deleted element is 30

Elements are

40

50

Deleted element is 40

Deleted element is 50

When tried to remove sixth element from empty queue

No elements to delete