CORE -JAVA-ASSIGNMENT-PART-1

**Part 1 Assignment Core java class 1:**

-1- Topics

1. Interview Questions:

* Example 1:

**package** part1SessionOne;

/\*REVERSE A GIVEN STRING ?

\* 1) create a string to be reversed and assign it to the variable.

\* 2) use decrimental for loop.

\* 3) use str.length() & str.charAt() methods.

\* 4) logic.

\* \*/

**public** **class** Ex1 {

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

String str = "pankaj sir academy";

**for** (**int** i = str.length() - 1; i >= 0; i--) {

// str.length() - 1 because index starts from zero

System.***out***.print(str.charAt(i));

}

}

}

* Example 2:

**package** part1SessionOne;

/\*COUNT THE NUMBER OF WORDS IN GIVEN STATEMENT ?\*/

**public** **class** Ex2 {

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

String str = "Hello my Name is Yog"; // bug is if there is space at beginning of

// sentence it will consider it as word too so we always trim first then we

// split based on spaces.

String[] s = str.trim().split(" "); // trim to trim first and last spaces.

System.***out***.println(s.length); // prints number of elements present in s array.

// \* to print words \*//

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

System.***out***.println(string);

}

}

}

* Example 3:

**package** part1SessionOne;

**import** java.util.Scanner;

/\*TO CHECK NUMBER OF OPENING AND CLOSING PARENTHESIS AND IF

\* SAME NUMBER PRINT NO ERROR IF NOT SAME PRINT ERROR ?\*/

**public** **class** Ex3 {

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

String condition = "yes";

**while** (condition.equals("yes")) {

Scanner scan = **new** Scanner(System.***in***); // scanner class

System.***out***.println("Enter the Paranthesis");

String str = scan.next(); // taking string as input

**int** count1 = 0;

**int** count2 = 0;

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

**if** (str.charAt(i) == '(') {

count1++;

} **else** **if** (str.charAt(i) == ')') {

count2++;

}

}

**if** (count1 == count2) {

System.***out***.println("NO ERROR");

} **else** {

System.***out***.println("ERROR");

}

System.***out***.println("do you want to continue? Yes /No");

condition = scan.next().toLowerCase();

}

}

}

* Example 4:

**package** part1SessionOne;

/\*BUILD A SQUARE 5x5\*/

/\*Note make sure all are "Print" and not "println"\*/

**public** **class** Ex4 {

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

**for**(**int** i=0; i<5; i++) {

**for**(**int** j =0; j<5; j++) {

System.***out***.print("\*");

}

System.***out***.print("\n");

}

}

}

* Example 5:

**package** part1SessionOne;

/\*BUILD A RIGHT ANGLE TRIANGLE\*/

/\*Note make sure all are "Print" and not "println"\*/

**public** **class** Ex5 {

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

**for** (**int** i = 0; i < 5; i++) {

**for** (**int** j = 0; j < 5; j++) {

**if** (i == 0 && j == 1 || i == 0 && j == 2 || i == 0 && j == 3 || i == 0 && j == 4 || i == 1 && j == 2

|| i == 1 && j == 3 || i == 1 && j == 4 || i == 2 && j == 3 || i == 2 && j == 4

|| i == 3 && j == 4) {

System.***out***.print(" ");

}**else** {

System.***out***.print("\*");

}

}

System.***out***.print("\n");

}

}

}

**Part 1 Assignment Core java class 2:**

-4- Topics:

1. Mutable and Immutable:

* Example 1:

**package** part1SesssionTwo;

/\*Mutable and Immutable

\* Mutable: Is something where in class object property keeps on changing.

\* Immutable: once its object is created then its state cannot be altered.

\* \*/

**public** **class** Ex1 {

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

}

}

1. Steps to create Immutable Class:

* Example 1:

**package** part1SesssionTwo;

/\*Steps to Create Immutable class:

\* 1) Create a final class

\* 2) Set the values of the variables/properties using only constructors.

\* 3) Make the properties as final.

\* 4) Do-not provide any setters for these properties, only provide getters.

\* \*/

**final** **public** **class** Ex2 {

**private** **final** **int** age;

**private** **final** String name;

**public** Ex2(**int** age, String name) { // Initializing properties values.

**this**.age = age;

**this**.name = name;

}

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

Ex2 ex = **new** Ex2(20, "pankaj");

// object creation to assign values by calling constructor.

}

**public** String getName() { // getters

**return** name;

}

**public** **int** getAge() { // getters

**return** age;

}

}

1. String Is Immutable:

* Example 1:

**package** part1SesssionTwo;

/\*STRING IS IMMUTABLE\*/

/\* Note:

\* 1)Even if we reassign value to String reference variable it won’t change instead its values there itself

\* instead it will create new object and store in it where as past object will go for

\* garbage collection so String class is IMMUTABLE.

\* 2) always use s1.equals(s2) to compare values of 2 Strings\*/

**public** **class** Ex3 {

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

String s1 = **new** String("xyz"); // object 1

String s2 = **new** String("xyz"); // object 2

System.***out***.println(s1 == s2); // compares object address.

System.***out***.println(s1.equals(s2));// values are compared special type only for string

}

}

* Example 2:

**package** part1SesssionTwo;

/\* $$VERY VERY IMPORTANT TOPICS$$

\*

\* ######################

\* String s1 = "pankaj"; //object 1 and stores pankaj in it.

\* String s2 = "pankaj"; // as values are same it will point to object 1 only

\* String s3 = "Pankaj"; // as java is case sensitive this will create new object2 and stores Pankaj.

\* ######################

\* String s1 = new String("pankaj");// creates a new object 1 and stores pankaj.

\* String s2 = "pankaj"; // even though s1 and s2 values are same still as s1 is manually created in new object so here it will create another object 2 and stores pankaj in it.

\* String s3 = "pankaj"; // as values are same it will point to object 2 as here no new object created manually.

\* String s4 = new String("pankaj"); // creates a new object 3 and stores pankaj.

\*

\* s1 == s4 -> false

\* s2 == s1 -> false

\* s2 == s3 -> true

\* s2 == s4 -> false

\* s3 == s4 -> false

\* s3 == s1 -> false

\*######################

\* \*/

**public** **class** Ex4 {

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

}

}

1. String Constant Pool:

* Example 1:

**package** part1SesssionTwo;

/\*STRING CONSTANT POOL: where object is created is stored in string constant pool.

\*

\* ######################

\* String s1 = "xyz"; // object 1 which is stored in String Constant Pool.

\* String s2 = "xyz"; // object 1 which is stored in String Constant Pool.

\* String s3 = "abc"; // object 2 which is stored in String Constant Pool.

\* String s4 = "abc"; // object 2 which is stored in String Constant Pool.

\* ######################

\* \*/

**public** **class** Ex5 {

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

}

}

**Part 1 Assignment Core java class 3:**

-1- Topics:

1. Interview Questions:

* Example 1:

**package** part1SessionThree;

/\*STRING TO UPPERCASE & LOWERCASE\*/

**public** **class** Ex1 {

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

String str = "pAnkaJ Sir ACAdemy";

System.***out***.println(str.toLowerCase()); // we have built in method to convert into lower case.

System.***out***.println(str.toUpperCase());// we have built in method to convert into upper case.

}

}

* Example 2:

**package** part1SessionThree;

/\*REMOVE WHITE SPACES\*/

**public** **class** Ex2 {

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

String str = " pankaj sir Acedemy ";

System.***out***.println(str);// string printed with first and last spaces.

System.***out***.println(str.trim()); // is used to trim first and last white spaces in given string

}

}

* Example 3:

**package** part1SessionThree;

/\*CHECK STRING IF IT STARTS/ENDS WITH ANY LETTER ?\*/

**public** **class** Ex3 {

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

String str = "aabccde";

System.***out***.println(str.startsWith("a")); // true as given string starts with a.

System.***out***.println(str.endsWith("e"));// true as given string ends with e.

System.***out***.println(str.startsWith("d"));// false as given string doesn't starts with d.

System.***out***.println(str.endsWith("f"));// false as given string ends starts with f.

}

}

* Example 4:

**package** part1SessionThree;

/\*TO CHECK LENGTH OF STRING\*/

**public** **class** Ex4 {

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

String str = "pankaj ";

System.***out***.println(str.length());// length of string white spaces included.

}

}

* Example 5:

**package** part1SessionThree;

/\*String.valueOf(variable) method converts given type such as integer,

\* float, double, boolean etc to string as shown below\*/

**public** **class** Ex5 {

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

**char**[] i = {'a','b','c'};

String str = String.*valueOf*(i);

System.***out***.println(str);

}

}

* Example 6:

**package** part1SessionThree;

/\*Assignment 1:

\* WRITE A PROGRAM TO FIND HOW MANY a's and b's are there in (aabaaaabaa)?\*/

**public** **class** Ex6 {

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

String str = "aabaaaabaa";

**int** a=0; // to count number of a's as number of times if condition for a runs its value keeps on increasing

**int** b =0; // to count number of b's as number of times else if condition for b runs its value keeps on increasing

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

**if**(str.charAt(i)=='a') {

a++; //

}**else** **if**(str.charAt(i)=='b') {

b++;

}

}

System.***out***.println("number of a's in given string are :"+a);

System.***out***.println("number of b's in given string are :"+b);

}

}

* Example 7:

**package** part1SessionThree;

/\*Assignment 2:

\* output:

\* 1

\* 23

\* 456

\* 78910

\* 1112131415

\* \*/

**public** **class** Ex7 {

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

**int** k = 1;// 2.......15

**for** (**int** r = 1; r <= 5; r++) { // 1 2 3 4 5

**for** (**int** c = 1; c <= r; c++) { // 1 2 3 4 5

System.***out***.print(k);

k++;

}

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

}

}

}

* Example 8:

**package** part1SessionThree;

/\*FIND DUPLICATE ELEMENTS IN ARRAY AND REMOVE IT ?\*/

**public** **class** Ex8 {

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

}

}

**Part 1 Assignment Core java class 4:**

-2- Topics:

1. Threads:

* Example 1:

**package** part1SessionFour;

/\*Threads:

\* 1) Multitasking done at program level is called as threads.

\* 2) the main purpose of thread is to improve the performance

\* of application by reducing execution time.

\* 3) Example file searching logic between 3 folders to search a file is

\* the best example.

\* \*/

**public** **class** Ex1 {

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

}

}

1. Thread:

* Example 1:

**package** part1SessionFour;

/\*A)Thread:

\* 1) is a class

\* 2) consists of 2 methods

\* a)->run(); --> we override this method and write what we want to multi-task with

\* main method.

\* b)->start(); --> we use this to start the threading operation when it is

\* appended with object reference variable (a1.start()).

\*

\* B) Runnable:

\* 1) its an interface.

\* 2) consist of only run() method.

\* \*/

**public** **class** Ex2 {

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

}

}

* Example 2:

**package** part1SessionFour;

/\*1st concept i.e., Thread\*/

/\*Note:

\* 1)make sure in main method first object should be created

\* and should call the start method and next we have to write different logic.

\*

\* 2)here in output we can see clearly that main method runs for some time

\* and run method runs for some time and it repeats and order it executes

\* is unpredictable.

\*

\* 3)main method is also considered as thread so here total of 2 threads are there.

\* \*/

**public** **class** Ex3 **extends** Thread {

@Override

**public** **void** run() { // eligible for multitasking

**for** (**int** i = 0; i < 1000; i++) {

System.***out***.println("run method");

}

}

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

// order 1

Ex3 ex = **new** Ex3();

ex.start();

// order 2

**for** (**int** i = 0; i < 1000; i++) {

System.***out***.println("main method");

}

// here in output we can see clearly that main method runs for some time

// and run method runs for some time and it repeats and order it executes

// is unpredictable.

}

}

* Example 3:

**package** part1SessionFour;

/\*2nd Example for Thread concept\*/

/\*total of 4 threads are there\*/

**public** **class** Ex4 **extends** Thread {

String name;

Ex4(String name) {

**this**.name = name;

}

@Override

**public** **void** run() { // eligible for multitasking

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println(**this**.name);

}

}

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

Ex4 ex = **new** Ex4("XXX");

Ex4 ex1 = **new** Ex4("YYY");

Ex4 ex2 = **new** Ex4("ZZZ");

ex.start(); // thread 1 starts

ex1.start(); // thread 2 starts

ex2.start(); // thread 3 starts

}

}

* Example 4:

**package** part1SessionFour;

/\* 2nd concept i.e., runnable \*/

/\*Note:

\* 1) here we don't have start() method by default so we

\* create object and also create thread class to get start() method.

\*

\* 2)here in output we can see clearly that main method runs for some time

\* and run method runs for some time and it repeats and order it executes

\* is unpredictable.

\*

\* 3)main method is also considered as thread so here total of 3 threads are there.

\* \*/

**public** **class** Ex5 **implements** Runnable { // run() only

String name;

Ex5(String name) {

**this**.name = name;

}

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 100; i++) {

System.***out***.println(**this**.name);

}

}

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

Ex5 ex = **new** Ex5("XXX"); // object to get reference variable.

Thread t1 = **new** Thread(ex); // Thread object to get start() method.

t1.start();

Ex5 ex1 = **new** Ex5("YYY"); // object to get reference variable.

Thread t2 = **new** Thread(ex1); // Thread object to get start() method.

t2.start();

**for** (**int** i = 0; i < 100; i++) {

System.***out***.println("main");

}

}

}

**Part 1 Assignment Core java class 5:**

-1- Topics:

1. Drawbacks of Thread:

* Example 1:

**package** part1SessionFive;

/\*Drawbacks of Thread:

\* 1) Bank where money is deposited from one end and withdrawn simenataneously on other

\* end without even complete full deposit where output will be corrupted so here

\* we wont use thread concept.

\* 2) 2 people booking only one remaining ticket at the exact same time even here data

\* will be corrupted, so we don't use thread concept.

\* 3) redeeming of same coupon.

\*/

**public** **class** Ex1 {

}

**Part 1 Assignment Core java class 6:**

-2- Topics:

1. Thread Synchronization:

* Example 1:

**package** part1SessionSix1;

/\*Thread Synchronization:

\* When 2 threads are operating on common data the data might get corrupted because of

\* multitasking , to make the thread operates one after another we use synchronized

\* keyword, the thread which is acquired the lock can only execute the block whereas

\* the other thread would be in wait states only when the first thread release the

\* lock the other thread will get opportunity and execute the lock.

\*/

**public** **class** Ex1 {

**int** balance =0;

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

Ex1 ex = **new** Ex1();

ex.account();

System.***out***.println(ex.balance);

}

**private** **void** account() {

Thread t1 = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

add();// this.add();

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

sub();// this.add();

}

});

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

}**catch**(Exception e) {

e.printStackTrace();

}

}

**private** **synchronized** **void** add() { //synchronized is used to make the add to complete first then go to subtract so that there wont be any data corruption.

**for**(**int** i=0; i<1000;i++) {

balance = balance+i;

}

}

**private** **synchronized** **void** sub() {

**for**(**int** i=0; i<1000;i++) {

balance = balance-i;

}

}

}

* Example 2:

**package** part1SessionSix1;

/\*Thread Synchronization:

\* When 2 threads are operating on common data the data might get corrupted because of

\* multitasking , to make the thread operates one after another we use synchronized

\* keyword, the thread which is acquired the lock can only execute the block whereas

\* the other thread would be in wait states only when the first thread release the

\* lock the other thread will get opportunity and execute the lock.

\*/

**public** **class** Ex2 **extends** Ex2\_1 {

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

Ex2 ex = **new** Ex2();

ex.start();

**synchronized** (ex) {

**try** {

ex.wait();

} **catch** (Exception e) {

e.printStackTrace();

}

System.***out***.println(*total*);

}

}

}

**package** part1SessionSix1;

**public** **class** Ex2\_1 **extends** Thread {

**static** **int** *total* = 0;

@Override

**public** **synchronized** **void** run() {

**for** (**int** i = 0; i < 1000; i++) {

*total* = *total* + i;

}

notify();

}

}

1. Thread Pool:

* Example 1:

**package** part1SessionSix1;

/\*Thread Pool:

\*

\* \*/

**public** **class** Ex3 **extends** Thread {

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

Ex3 ex = **new** Ex3();

System.***out***.println(ex.getState());

ex.start();

**try** {

Thread.*sleep*(5000);

} **catch** (Exception e) {

e.printStackTrace();

}

System.***out***.println(ex.getState());

}

**public** **void** run () {

System.***out***.println("test");

}

}

**Part 1 Assignment Core java class 7:**

-3- Topics:

1. Thread Priority:

* Example 1:

**package** part1SessionSeven;

/\*Thread Priority:

\* 1) it decides which thread runs first and which thread runs later.\

\* 2) if we set the priority then it is the request made to thread scheduler there is

\* no assuring that it will be processed of approved.

\* 3) the minimum thread priority is 1 and maximum thread priority is 10, normal

\* thread priority is 5, however we can set thread priority with number between 1-10.

\*

\* \*/

**public** **class** Ex1 **extends** Thread {

@Override

**public** **void** run() {

System.***out***.println(**this**.name);

}

String name;

**public** Ex1(String name) {

**this**.name= name;

}

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

Ex1 ex = **new** Ex1("XXX");

Ex1 ex1 = **new** Ex1("YYY");

ex.setPriority(10); // with highest priority will run first.

ex1.setPriority(1);

System.***out***.println(ex.getPriority());

System.***out***.println(ex1.getPriority());

ex.start();

ex1.start();

}

}

1. Setting and getting name of thread:

* Example 1:

**package** part1SessionSeven;

/\*Setting and getting name of thread:

\* \*/

**public** **class** Ex2 **extends** Thread {

String name;

**public** Ex2(String name) {

**this**.name = name;

}

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

Ex2 ex = **new** Ex2("XXX");

Ex2 ex1 = **new** Ex2("YYY");

ex.setName("add amount");

ex1.setName("withdraw amount");

System.***out***.println(ex.getName());

System.***out***.println(ex1.getName());

ex.start();

ex1.start();

}

@Override

**public** **void** run() {

System.***out***.println(**this**.name);

}

}

1. Thread Pool (Theory):

* Example 1:

**package** part1SessionSeven;

/\*

\* Thread Pool:

\* -------------

\* 1)when you need to limit the number of threads running your application at same time

\* , this will help us to improve the performance of the application,i.e., instead

\* of starting new thread for every task execute currently task can be passed to

\* a thread pool.

\*

\* 2)Thread pool contains collection of thread, as soon as pool has any ideal thread

\* task is assigned to one then and gets executed.

\*

\* 3) Thread pool are often used in servers, each connection arriving from network

\* is wrapped as task and passed to thread pool.

\*

\* 4)The thread in the thread pool will process the request on the connections done

\* currently, this is how we can use existing threads instead of creating a new

\* thread and there by improve the performance in-turns of execution.

\*

\* \*/

**public** **class** Ex3 {

}

**Part 1 Assignment Core java class 8:**

-4- Topics:

1. Enum:

* Example 1:

**package** p1;

/\*

\* Enum:

\* 1) enum is a collection of constants.

\*

\* \*/

**public** **enum** Ex1 {

***jan***, ***feb*** , ***march***, ***april***, ***may***, ***june***, ***july*** , ***aug***, ***sept***, ***oct*** , ***nov*** , ***dec***;

}

**package** p1;

**public** **class** Ex1\_1 {

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

System.***out***.println(Ex1.***april***); // here we can only access constants created enum

// class other than that we can't access anything

}

}

* Example 2:

**package** p1;

/\*

\* Enum Example 2 :

\* \*/

**public** **enum** Ex2 {

***Mr***, ***Mrs***, ***Prof***,***Dr***;

}

**package** p1;

// continuation of Ex2

**public** **class** Ex2\_1 {

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

System.***out***.println(Ex2.***Dr***);

}

}

1. Wrapper class:

* Example 1:

**package** p1;

/\*

\* Wrapper class:

\* 1) Here the values are stored in the object.

\* 2) The process of storing the value inside an object is called as wrapping or boxing.

\* 3)Reading the value from the object is called as unboxing.

\* \*/

**public** **class** Ex3 {

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

Integer i = 10; // or Integer i = new Integer(10);

System.***out***.println(Integer.***MAX\_VALUE***); // gives max value of integer.

System.***out***.println(i.longValue()); // converts integer into long.

System.***out***.println(i.hashCode()); // converts integer into hash code

System.***out***.println(Integer.***SIZE***); // gives size of the Integer i.e, 4 bytes--> 32bits(1byte=4bits)

System.***out***.println(i.toString());// converts integer into string

System.***out***.println(i.doubleValue());// converts integer into double

System.***out***.println(i.byteValue());// converts integer into byte value.

}

}

1. Types of Wrapper class:

* Example 1:

**package** p1;

/\*

\* Type of wrapper class:

\* 1) initializing Wrapper classes is mandatory.

\* \*/

**public** **class** Ex4 {

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

Byte b = 34;

Short s = 20;

Integer i = 30;

Long l = 40l;

Float f = 50.8f;

Double d = 70.89;

Character c = 'a';

Boolean o = **true**;

System.***out***.println(b);

System.***out***.println(s);

System.***out***.println(i);

System.***out***.println(l);

System.***out***.println(f);

System.***out***.println(d);

System.***out***.println(c);

System.***out***.println(o);

}

}

1. Finalize:

* Example 1:

**package** p1;

/\*

\* Finalize:

\* ---------

\* 1) is a method present inside object class.

\* 2) garbage collection logic is implemented in Finalize method.

\* \*/

**public** **class** Ex5 **extends** Object {

**protected** **void** finalize() {

System.***out***.println(1000);

}

@SuppressWarnings("unused")

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

Ex4 ex = **new** Ex4();

ex= **null**;

System.*gc*(); // it will call GC and will print anything inside finalize block.

// Note: if we dont make ex= null then there is no concept of garbage collection.

}

}

**Part 2 Assignment Core java class 1:**

-2- Topics:

1. Throws Keyword:

* Example 1:

package p1;

import java.io.FileWriter;

/\*Throw and Throws:

\* -----------------

\* A) Throws keyword

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 1)it is applied on a method.

\* 2)if any exception occurs in a method then exception will be passed on

\* to calling statement of the method.

\* \*/

public class Ex1 {

public static void main(String[] args) {

Ex1 ex = new Ex1();

/\*UNHANDELED EXCEPTION\*/

//ex.test();

/\*HANDELED EXCEPTION\*/

try {

ex.test(); // exception will throw to the calling state it should be surrounded in try

// catch block

} catch (Exception e) {

e.printStackTrace();

}

}

@SuppressWarnings("resource")

public void test() throws Exception { // can be only applied on a method

@SuppressWarnings("unused")

FileWriter fw = new FileWriter("D://test.txt");

}

}

* Example 2:

package p1;

import java.io.FileWriter;

/\*2nd way to avoid exception\*/

public class Ex2 {

public static void main(String[] args) throws Exception { // 2nd way

Ex1 ex = new Ex1();

ex.test(); // exception will throw to the calling state it should be surrounded in try

// catch block

}

@SuppressWarnings("resource")

public void test() throws Exception { // can be only applied on a method

@SuppressWarnings("unused")

FileWriter fw = new FileWriter("D://test.txt");

}

}

* Example 3:

package p1;

import java.io.FileWriter;

import java.io.IOException;

import java.sql.DriverManager;

import java.sql.SQLException;

/\* Note:

\* after Throws keyword we can give multiple class Names.

\* \*/

public class Ex3 {

@SuppressWarnings("resource")

public static void main(String[] args) throws IOException, SQLException {

@SuppressWarnings("unused")

FileWriter fw = new FileWriter("D://test.txt");

DriverManager.getConnection("","","");

}

}

1. Throw keyword:

* Example 4:

package p1;

/\*Throw and Throws:

\* -----------------

\* B) Throw keyword

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 1) It helps us to create customized Exception as per the requirements of the developer

\* 2) example :

\* \*/

/\*

\* Creating own exceptions first example(user defined Exception):

\* \*/

@SuppressWarnings("serial")

public class InsufficientFunds extends Throwable {

// any class to create customized exception we have to inherit throwable.

}

**package** p1;

**import** java.util.Scanner;

**public** **class** A {

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

**int** balance = 500;

System.***out***.println("Enter amount to withdraw: ");

@SuppressWarnings("resource")

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

**int** amount = s.nextInt();

**if**(balance>amount) {

System.***out***.println("please collect the cash");

}**else** {

**try** {

**throw** **new** InsufficientFunds();

}**catch** (InsufficientFunds e)/\*or just use throws InsufficientFunds side to main method.\*/ {

e.printStackTrace();

System.***out***.println("Low Balance");

}

}

}

}

* Example 2:

**package** p1;

/\*

\* 2nd Example on throw

\* \*/

@SuppressWarnings("serial")

**public** **class** ApplicationCrashed **extends** Throwable{

}

**package** p1;

**public** **class** B {

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

**try** {

**throw** **new** ApplicationCrashed();

} **catch** (ApplicationCrashed e) {

System.***out***.println("application restarting");

}

}

}

**Part 2 Assignment Core java class 2:**

-1- Topics:

1. Regular Expression:

* Example 1:

package p1;

/\*\*

\* Regular Expression:

\* 1)(refer Ex2)-> Pattern p =Pattern.compile("[abc]"); //what we want to search.

\* p.matcher("a6b#@z9D");

\* while(m.find()){

\* sop(m.start()+"...."+m.group());

\* }

\*

\* output:

\* a....0

\* b....2

\* z....5

\*/

public class Ex1 {

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

}

}

* Example 2:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Example 1 for Regular Expression.

\*/

public class Ex2 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[abz]"); // whatever in square brackets will be searched for it.

Matcher m = p.matcher("a6b#@z9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 3:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Example 2: for all a-z alphabets.

\*/

public class Ex3 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[a-z]"); // here it is searching all the words from a-z lower cases.

Matcher m = p.matcher("a6b#@z9DuGHASUgfiu"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 4:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Example 3: for all numbers.

\*/

public class Ex4 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[0-9]"); // here it is searching all the numbers from 0-9.

Matcher m = p.matcher("a6b#@z9154gawSSERgs45r3"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

* Example 5:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\*Example 4: for all A-Z in upper-case alphabets.

\*/

public class Ex5 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[A-Z]"); // here it is searching all the alphabets from A-Z all upper-cases.

Matcher m = p.matcher("a6b#@z9154gawSSERgs45r3"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

* Example 6:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\*Example 4: for all A-Z in upper-case & upper-case alphabets and also digits.

\*/

public class Ex6 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[a-zA-Z0-9]"); // here it is searching all the alphabets from A-Z all upper&lower-case and also all the digits from 0-9.

Matcher m = p.matcher("a6b#@z9154gawSSERgs45r3"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

* Example 7:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\*Example 4: for anything other than lower-case alphabets.

\*/

public class Ex7 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[^a-z]"); // here it will search everything other than lower-case alphabets.

Matcher m = p.matcher("a6b#@z9154gawSSERgs45r3"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

* Example 8:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\*Example 4: for only special cases.

\*/

public class Ex8 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[^a-zA-Z0-9]"); // here it is searching anything other than all the alphabets from A-Z all upper&lower-case and also all the digits from 0-9.

Matcher m = p.matcher("a6b#@z9154gawSSERgs45r3"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

* Example 9:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Note:

\* Anything we write in {} will form the number of groups of given string as shown in example below.

\*/

public class Ex9 {

public static void main(String[] args) {

Pattern p = Pattern.compile("[0-9]{2}"); //Anything we write in {} will form the number of groups of given string -->if 2 then 5 groups , if 5 then 2 groups if 11 then 0 groups.

Matcher m = p.matcher("6361317502"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start() + "...." + m.group());

// m.start() will print the index of matching letter and m.group() will print

// the letter of given index.

}

}

}

**Part 2 Assignment Core java class 3&4:**

-1- Topics:

1. More on Regular Expression:

* Example 1:

package p1;

/\*\*

\* When and where to use all this in regular expression.

\* "//s" -> //S --> //d --> //D

\*/

public class Ex1 {

}

* Example 2:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* How to get white spaces in given string using Regular Expression.

\*/

public class Ex2 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\s"); // give me white space from given string.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 3:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* upper-case S gives anything other than white spaces.

\*/

public class Ex3 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\S"); //upper-case S gives anything other than white spaces.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 4:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* lower-case d gives every numbers/digits and only digits.

\*/

public class Ex4 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\d"); //lower-case d gives every numbers/digits.

Matcher m = p.matcher("a6b#@z9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 5:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* upper-case D gives everything other than numbers/digits even it will give white spaces.

\*/

public class Ex5 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\D"); //upper-case D gives everything other than numbers/digits.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 6:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* gives everything other than special characters and not even white spaces other than that everything.

\*/

public class Ex6 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\w"); //gives everything other than special characters and not even white spaces other than that everything.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 7:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* gives special characters and even white spaces other than that everything will be not printed.

\*/

public class Ex7 {

public static void main(String[] args) {

Pattern p = Pattern.compile("\\W"); //gives special characters and even white spaces other than that everything will be not printed.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

System.out.println(m.start()+"...."+m.group());

//m.start() will print the index of matching letter and m.group() will print the letter of given index.

}

}

}

* Example 8:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* write a program to validate a given field where input should not contain any white space using

\* regular Expression.

\*/

public class Ex8 {

static int count=0;

public static void main(String[] args) {

Pattern p = Pattern.compile("\\s"); // give me white space from given string.

Matcher m = p.matcher("a6b #@z 9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

count++;

}

if(count!=0) {

System.out.println("error");

}else {

System.out.println("input accepted");

}

}

}

* Example 9:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* write a program to validate a given field where input should not contain any special characters using

\* regular Expression.

\*/

public class Ex9 {

static int count=0;

public static void main(String[] args) {

Pattern p = Pattern.compile("\\W"); // give me white space from given string.

Matcher m = p.matcher("a6bz9D"); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

count++;

}

if(count!=0) {

System.out.println("error");

}else {

System.out.println("input accepted");

}

}

}

* Example 10:

package p1;

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Validating Name using Regular Expression.

\*/

public class Ex10 {

static int count=0;

@SuppressWarnings("resource")

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter Your Name: ");

String str = scan.nextLine();

Pattern p = Pattern.compile("[^a-zA-Z]"); // accepts anything other than name with lower and upper case.

Matcher m = p.matcher(str); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

count++;

}

if(count!=0 || str.length()<3) {

System.out.println("Error");

}else {

System.out.println("Input Accepted");

}

}

}

* Example 11:

package p1;

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Note: not correct many bugs.

\* Validating mobile phone using Regular Expression so many bugs.

\*/

// matcher takes only string as an input

public class Ex11 {

static int count = 0;

@SuppressWarnings("resource")

public static void main(String[] args) {

try {

Scanner scan = new Scanner(System.in);

System.out.println("Enter Your Mobile number: ");

Long mobileNumber = scan.nextLong();

String str = mobileNumber.toString();

Pattern p = Pattern.compile("[6-9][0-9]{9}"); // accepts anything other than name with lower and upper case.

Matcher m = p.matcher(str); // it is searching in this give string inside matcher method.

while (m.find()) { // it will help us to find the given words in step 1.

count++;

}

if (count != 0 || str.length() < 10) {

System.out.println("Error");

} else if(str.length()==10){

System.out.println("Input Accepted");

}

} catch (Exception e) {

System.out.println("Invalid Input");

}

}

}

* Example 12:

package p1;

**import** java.util.Scanner;

/\*\*

\* easy way to validate mobile number using matches method.

\*/

public class Ex12 {

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

**try** (Scanner scan = **new** Scanner(System.***in***)) {

System.***out***.println("Enter Your Mobile number: ");

String str = scan.next();

String regex="[6-9][0-9]{9}";

**if**(str.matches(regex)) {

System.***out***.println("valid");

}**else** {

System.***out***.println("invalid");

}

}

}

}

**Part 2 Assignment Core java class 5:**

-1- Topics:

1. Wild cards in Regular Expression :

* Example 1:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*WILD CARDS IN REGULAR EXPRESSION : char\*, char+, char?\*/

/\*\*

\* char\*-> it gives us zero occurrences or group of occurrences of particular

\* character.

\*/

public class Ex1 {

public static void main(String[] args) {

Pattern p = Pattern.compile("a\*"); // group of a's or zero occurrences of a.

Matcher m = p.matcher("abaababababab");

while (m.find()) {

System.out.println(m.start() + "......." + m.group());

}

}

}

* Example 2:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* char+-> group of occurrences of a and not any zero occurrence of a.

\*/

public class Ex2 {

public static void main(String[] args) {

Pattern p = Pattern.compile("a+"); // group of occurrences of a and not any zero occurrence of a.

Matcher m = p.matcher("abaababababab");

while (m.find()) {

System.out.println(m.start() + "......." + m.group());

}

}

}

* Example 3:

package p1;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* char? -> only single and zero occurrences of 'a' and not any group occurrence of 'a'.

\*/

public class Ex3 {

public static void main(String[] args) {

Pattern p = Pattern.compile("a?"); //only single and zero occurrences of 'a' and not any group occurrence of 'a'.

Matcher m = p.matcher("abaababababab");

while (m.find()) {

System.out.println(m.start() + "......." + m.group());

}

}

}

* Example 4:

package p1;

import java.util.StringTokenizer;

/\*\*

\* StringTokenizer:

\* 1) Major importance is to split the string.

\*/

/\*Write a program to count and split the sentence into words based on white

\* spaces of any type using StringTokenizer, instead of using split() use this so that no bugs are present.

\* \*/

public class Ex4 {

public static void main(String[] args) {

int count=0;

StringTokenizer str = new StringTokenizer(" pankaj sir academy ");

while (str.hasMoreTokens()) {

System.out.println(str.nextToken());

count++;

}

System.out.println(count);

}

}

* Example 5:

package p1;

**import** java.util.StringTokenizer;

/\*\*

\* split given date/string based on delimiter using StringTokenizer.

\*/

public class Ex5 {

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

**int** count=0;

StringTokenizer str = **new** StringTokenizer("12-12-2020","-"); //split based on delimiter.

**while** (str.hasMoreTokens()) {

System.***out***.println(str.nextToken());

count++;

}

System.***out***.println(count);

}

}

* Example 6:

package p1;

**import** java.util.regex.Pattern;

/\*\*

\* Note:Homework

\* Using Regular Expression write a program to split the given string based on spaces.

\*/

public class Ex6 {

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

**int** count=0;

Pattern p = Pattern.*compile*("\\s");

String[] split = p.split("pankaj sir academy");

**for** (String string : split) {

System.***out***.println(string);

count++;

}

System.***out***.println(count);

// Matcher m = p.matcher("pankaj sir academy"); // here changes to be done is p.something

// while(m.find()) {

// System.out.println(m.start());

// }

}

}

* Example 7:

package p1;

/\*\*

\* Cloning:The process of creating a replica of a particular object by copying

\* the content of one object completely into another object.

\*/

public class Ex7 implements Cloneable { // Clone-able is an marker interface.

public static void main(String[] args) throws Exception {

Ex7 ex = new Ex7();

System.*out*.println(ex);

Ex7 ex1 = (Ex7) ex.clone(); // different object cloned object

System.*out*.println(ex1);

}

}

* Example 8:

package p1;

/\*\*

\* HASH CODE IN JAVA: will return integer representation of memory address as shown below.

\*/

public class Ex8 {

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

Ex8 ex8 = **new** Ex8();

System.***out***.println(ex8.hashCode()); // converts hexadecimal values into integer representation.

}

}

**Part 2 Assignment Core java class 6:**

-1- Topics:

1. Annotations:

* Example 1:

package p1;

/\*\*

\* Annotations: communication to java complier to do something/command the compiler.

\* \*\*\*\*\*\*\*\*\*\*\*\*

\* 3 Types:

\* --------

\* 1)@Override

\* 2)@SupressWarning

\* 3)@Depricated

\*

\*/

public class Ex1 {

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

}

}

* Example 2:

package p1;

/\*\*

\* **@SupressWarning**: me instructing compiler to tell to suppress the warning

\*/

public class Ex2 {

@SuppressWarnings("unused") // anything in method which gives warning will be suppressed

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

// @SuppressWarnings("unused") // only for one variable.

**int** i=10;

**int** j=10;

**int** g=10;

}

}

* Example 3:

package p1;

/\*\*

\* **@deprecated** : it tells which features are out-dated.

\*/

public class ~~Ex3~~ {

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

~~Ex3~~ ex3 = **new** ~~Ex3~~();

ex3.~~findById~~();

}

@Deprecated

**public** **void** ~~findById~~() {

System.***out***.println("i am searching record");

}

}

**Part 2 Assignment Core java class 7&8&9:**

-3- Topics:

1. MySql:

* Example 1:

package p1;

/\*My-SQL:

\* Refer Core Java Sessions.

\* \*/

//Connecting With MYSQL Database.

public class Ex1 {

}

* Example 2:

package p1;

// inserting data into SQL using JDBC

public class Ex2 {

}

* Example 3:

package p1;

//Eight Class:

/\*

\*

\* \*/

public class Ex3 {

}

* Example 4:

package p1;

//Ninth Class: refer core java sessions.

//CRUD operations:

/\* Example 1--Registering Data:

\* scanner class and use it to register by SQL Query-->'"+ variable +"'

\* Example 2--Deleting Data:

\* scanner class and use it to delete by SQL Query-->'"+ variable +"'

\* Example 3--Update Data:

\* scanner class and use it to update by SQL Query-->'"+ variable +"'

\* Example 3--Retrieve data

\* Retrieve data by SQL Query-->ResultSet set=executeQuery(select \* from employee );

\* use while and write printWriter and use while loop .next(),

\* and set.grtString(column number);

\* \*/

public class Ex4 {

}

**Part 2 Assignment Core java class 10:**

-4- Topics:

1. Collections:

* Example 1:

package p1;

/\*

\* Collections:

\* 1) stores group of object in it.

\* 2) in java collection is a framework which has readily

\* available logic to deal with different Data Structures.

\* --------------------------------------------------------

\* Types of Collections:

\* ####################

\* 1) LIST: its an interface.

\* a)ArrayList:

\* <\*> internally it is implemented as dynamic array.

\* <\*> initial size of ArrayList is 10.

\* <\*> when we exceed the initial size automatically arrayList size increases by 1.5 times.

\* <\*> ArrayList maintains insertion order.

\* <\*> it can consists of duplicate elements.

\* <\*> the advantage of array list is reading of data would give us best performance.

\* <\*> disadvantage, insertion of data in b/w of the array list will result in worst performance,

\* as the shuffle and data adjustments has to be done so performance error.

\* <\*> we can store heterogeneous data but it is not a good practice, so we use generic here.

\* b)Vector:

\* <\*>vector is not used much as it gives bad performance as all the methods are synchronized,

\* and due to this thread operations happens one after another therefore less performance.

\* c)LinkedList

\* (i)singly linked list:

\* (ii)Doubly linked list:

\*

\* \*/

public class Ex1 {

}

1. ArrayList:

* Example 1:

package p1;

**import** java.util.ArrayList;

//ArrayList

public class Ex2 {

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

ArrayList x = **new** ArrayList();

x.add(10); // add() method is present in arrayList to add values.

x.add(20); // it is also called as boxing as here objects are created in each add step not value.

x.add(**new** Integer(50)); // it is also same.

System.***out***.println(x);

}

}

* Example2:

package p1;

import java.util.ArrayList;

import java.util.Iterator;

//ArrayList

public class Ex3 {

public static void main(String[] args) {

ArrayList<Integer> x = new ArrayList<Integer>(); // using generic to store only integer values.

//1)Adding object-values.

x.add(10);

x.add(80); // maintains insertion order.

x.add(15);

x.add(10); //duplicate elements can be present.

System.out.println("add() method:" + x);

//2)adding object-values in between

x.add(1,500);//x.add(index,value);

System.out.println("x.add(index,value):"+x);

//3)adding 2 Arrays

ArrayList<Integer> y = new ArrayList<Integer>();

y.add(900);

y.add(850);

x.addAll(2, y);//x.addAll(index,collection/array);

System.out.println("x.addAll(index,collection/array):"+x);

//4)to search weather it contains or not.

if(x.contains(/\*object\*/500)) {

System.out.println("yes, it contains");

}else {

System.out.println("no, it doesnot contains");

}

//5)removing value based on index

x.remove(1); //x.remove(index of object);

System.out.println("x.remove(index of object):"+x);

//6)size of given ArrayList

int size = x.size();

System.out.println("x.size():"+size);

//7)read data based on index number

System.out.println(x.get(2));// x.get(index number of values to be fetched);

//8)read data using iterator

Iterator<Integer> itr = x.iterator();

while(itr.hasNext()) {

System.out.println("value of x :"+ itr.next());

}

}

}

1. Linked List:

* Example 1:

package p1;

/\*

\* LinkedList:

\* ----------

\* (i)singly linked list:

\* <\*> disadvantage is takes more memory.

\* <\*> we can understand that we reached last data by reading null value at end.

\* <\*> this is in one direction data flow i.e., head to tail, so it is called singly linked list.

\* <\*> adding data at beginning performance is good, but at the end and in between if we

\* add data then performance is very bad.

\* (ii)Doubly linked list:

\* <\*> 2 direction reading of data we can do.

\* <\*> disadvantage is more data as each node has 3 fields to fill.

\* <\*> adding data at end and beginning performance is good , but if data is

\* added in between then performance is bad.

\*

\* Note:

\* ----

\* <\*>in JDK linked list is internally implemented as doubly linked list.

\* <\*> array list and linked list both can consist of duplicate data in them.

\* \*/

public class Ex4 {

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

}

}

* Example 2:

package p1;

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

//linked list

public class Ex5 {

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

List<Integer> x = **new** LinkedList<Integer>();

/\*linked list also has same methods as array list but there are

some differences and some exclusive methods\*/

// using generic to store only integer values.

// 1)Adding object-values.

x.add(10);

x.add(80); // maintains insertion order.

x.add(15);

x.add(10); // duplicate elements can be present.

System.***out***.println("add() method:" + x);

// 2)adding object-values in between

x.add(1, 500);// x.add(index,value);

System.***out***.println("x.add(index,value):" + x);

// 3)adding 2 Arrays

ArrayList<Integer> y = **new** ArrayList<Integer>();

y.add(900);

y.add(850);

x.addAll(2, y);// x.addAll(index,collection/array);

System.***out***.println("x.addAll(index,collection/array):" + x);

// 4)to search weather it contains or not.

**if** (x.contains(/\* object \*/500)) {

System.***out***.println("yes, it contains");

} **else** {

System.***out***.println("no, it doesnot contains");

}

// 5)removing value based on index

x.remove(1); // x.remove(index of object);

System.***out***.println("x.remove(index of object):" + x);

// 6)size of given ArrayList

**int** size = x.size();

System.***out***.println("x.size():" + size);

// 7)read data based on index number

System.***out***.println(x.get(2));// x.get(index number of values to be fetched);

// 8)read data using iterator

Iterator<Integer> itr = x.iterator();

**while** (itr.hasNext()) {

System.***out***.println("value of x :" + itr.next());

}

}

}

* Example 3:

package p1;

//Example for Linked List

//Linked List

//part-1

**public** **class** Employee {

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**private** String firstName;

**private** String lastName;

private int id;

**public** Employee(String firstName,String lastName,**int** id) {

**this**.firstName=firstName;

**this**.lastName=lastName;

**this**.id=id;

}

}

* Example 4:

package p1;

**import** java.util.LinkedList;

//LinkedList

//part-2

public class B {

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

//create object and store values in it.

Employee arun = **new** Employee("arun","k",100);

Employee suresh = **new** Employee("suresh","m",101);

Employee ravi = **new** Employee("ravi","kumar",102);

LinkedList<Employee> employee = **new** LinkedList<>();

// using generic with employee class so that only we can store objects of it only

employee.add(arun);

employee.add(suresh);

employee.add(ravi);

**for** (Employee employee2 : employee) {

System.***out***.println(employee2.getFirstName());

System.***out***.println(employee2.getLastName());

System.***out***.println(employee2.getId());

System.***out***.println("--------------");

}

}

}

* Example 5:

package p1;

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

//linkedList

public class Ex6 {

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

LinkedList<Integer> x = **new** LinkedList<>();

// addFirst() --> read theory for explanation.

// every times a new element is added to first it becomes first element i.e., if

// 20 and 30 are added respectively then 30,20 in this way it is stored.

x.add(10);// always will be the last object as addFirst is in combination with it.

x.addFirst(100);

x.addFirst(200);

x.addFirst(300);

x.addFirst(400);

x.addFirst(200);

System.***out***.println(x);

//addLast()-->read theory for explanation.

// every times a new element is added to last it becomes last element i.e., if

// 20 and 30 are added respectively then 20,30 in this way it is stored.

x.add(5);// always will be the first object as addLast is in combination with it.

x.addLast(5000);

x.addLast(8000);

x.addLast(900);

x.addLast(50);

System.***out***.println(x);

}

}

1. Hash table:

* Example 1:

package p1;

/\* Hash Table:

\* -----------

\* Theory refer notes.\*/

/\* flow diagram of collection interface:

\* -----------------------------------

\* refer notes for theory.

\* \*/

public class Ex7 {

}

* Example 2:

package p1;

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

//HashTable

public class Ex12 {

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

Hashtable<Integer,String> student = **new** Hashtable<>(); //class up-casting

student.put(100, "pankaj");

student.put(101, "disha");

student.put(102, "pooja");

System.***out***.println(student);

System.***out***.println(student.values());

System.***out***.println(student.keySet());

System.***out***.println(student.get(102));

}

}

1. Hash set:

* Example 1:

package p1;

**import** java.util.HashSet;

//Hash set

public class Ex8 {

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

HashSet<Integer> hashset = **new** HashSet<>();

hashset.add(20);

hashset.add(30);

hashset.add(50);

hashset.add(80);

hashset.add(**null**);

System.***out***.println(hashset);

}

}

1. Linked Hash Set :

* Example 1:

package p1;

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

//Linked Hash Set

public class Ex9 {

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

LinkedHashSet<Integer> hashset = **new** LinkedHashSet<>();

hashset.add(20);

hashset.add(30);

hashset.add(50);

hashset.add(80);

hashset.add(**null**);

System.***out***.println(hashset);

}

}

1. Tree Set

* Example 1:

package p1;

**import** java.util.TreeSet;

//Tree Set

public class Ex10 {

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

TreeSet<Integer> treeset = **new** TreeSet<>();

treeset.add(20);

treeset.add(80);

treeset.add(30);

treeset.add(50);

// treeset.add(null); error

System.***out***.println(treeset);

}

}

1. Hash Map:

* Example 1:

package p1;

import java.util.HashMap;

import java.util.Map;

//HashMap

public class Ex11 {

public static void main(String[] args) {

Map<Integer,String> student = new HashMap<>(); //class up-casting

student.put(100, "pankaj");

student.put(101, "disha");

student.put(102, "pooja");

System.out.println(student);

System.out.println(student.values());

System.out.println(student.keySet());

System.out.println(student.get(102));

}

}

1. Comparator:

* Example 1:

package p1;

/\*comparator:

\* 1) it is used to order objects of user defined classes, if in sorting if obj1 comes first

\* and then obj2 then it will return -ve values .but if we sorting obj2 comes first and then

\* obj1 , then it will return +ve value, if both objects are same then it will return zero.

\*

\* ----------------------------------------------------------------------------------------

\* 3-Examples:

\* \*/

public class Ex13 {

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

//-ve

String x= "xyz"; // it holds good for custom objects too not only on inbuilt objects.

String y= "abc";

System.***out***.println(x.compareTo(y));

//+ve

String a= "abc"; // it holds good for custom objects too not only on inbuilt objects.

String b= "xyz";

System.***out***.println(a.compareTo(b));

//zero

String c= "abc"; // it holds good for custom objects too not only on inbuilt objects.

String d= "abc";

System.***out***.println(c.compareTo(d));

}

}

* Example 2:

package p1;

//Example on comparator.

public class Ex14 {

**int** id;

String name;

**public** Ex14(**int** id, String name) {

**this**.id=id;

**this**.name=name;

}

}

package p1;

**import** java.util.Comparator;

//continuation of EX14

**public** **class** Ex14\_1 **implements** Comparator<Ex14>{

@Override

**public** **int** compare(Ex14 o1, Ex14 o2) {

**return** o1.name.compareTo(o2.name);

}

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

Ex14\_1 ex = **new** Ex14\_1();

**int** val = ex.compare(**new** Ex14(100,"abc"),**new** Ex14(101,"xyz"));

System.***out***.println(val);

}

}

1. Generics:

* Example 1:

package p1;

/\*Generics:

\* gives us the flexibility to dynamically allocate the data type in program.

\* \*/

//Example 1:

**public** **class** Ex15<T> {

T val;

Ex15(T x){

**this**.val=x;

System.***out***.println(val);

}

}

-----------------

package p1;

/\*Generic:

\* gives us the flexibility to dynamically allocate the data type in program.

\* \*/

//Example 1 continuation:

public class Ex15\_1 {

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

Ex15<String> a1 = **new** Ex15<>("pankaj");

Ex15<Integer> a2 = **new** Ex15<>(50);

Ex15<Character> a3 = **new** Ex15<>('j');

}

}

1. J-Shell:

* Example 1:

package p1;

/\*J-shell:

\* JDK9 a new feature J-shell was added that helps us

\* to quickly build java code in it, to access J-shell go to java jdk,

\* bin , then double click on jshell.exe file.

\* \*/

/\*NOTE:

\* program of writing uni-code directly. jdk 9 and above\*/

public class Ex16 {

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

System.***out***.println("\u265A");

System.***out***.println("\u265B");

System.***out***.println("\u265C");

}

}