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:**

-4- Topics: