1. **Write the class Date having attributes like day, month & year. Add default & parameterized constructors. Add getters & setters. Add method to print the date. Add method to swap two dates.**

**package** com.zensar;

**public** **class** QuesDate {

**private** String day;

**private** String month;

**private** String year;

QuesDate()

{

**this**.day="Wednesday";

**this**.month="february";

**this**.year="2022";

}

QuesDate(String day,String month,String year)

{

**this**.day=day;

**this**.month=month;

**this**.year=year;

}

**public** **void** setDay(String day) {

**this**.day = day;

}

**public** **void** setmonth(String month) {

**this**.month = month;

}

**public** **void** setyear(String year) {

**this**.year = year;

}

**public** String getDay() {

**return** day;

}

**public** String getMonth() {

**return** month;

}

**public** String getyear() {

**return** year;

}

**public** **static** **void** printDate() {

QuesDate quesdate = **new** QuesDate();

String day = quesdate.getDay();

String month = quesdate.getMonth();

String year = quesdate.getyear();

System.***out***.println("Date : " + day + "/" + month + "/" + year);

}

**public** **static** **void** swapDate() {

QuesDate quesdate = **new** QuesDate();

String day = quesdate.getDay();

String month = quesdate.getMonth();

String year = quesdate.getyear();

quesdate.setDay("thursday");

String day2 = quesdate.getDay();

System.***out***.println(" before swapping = " + day + " " + day2);

System.***out***.println("Date : " + day + "/" + month + "/" + year);

String temp;

temp = day;

day = day2;

day2 = temp;

System.***out***.println(" after swapping =" + day + " " + day2);

System.***out***.println("Date : " + day + "/" + month + "/" + year);

}

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

*printDate*();

*swapDate*();

}

}

Output:

Date : Wednesday/february/2022

before swapping = Wednesday thursday

Date : Wednesday/february/2022

after swapping =thursday Wednesday

Date : thursday/february/2022

1. **Write a class ComplexNumber having attributes real & imaginary. Add functions like add, subtract, multiply & division.**

**package** com.zensar;

**public** **class** ComplexNumbers {

**double** real, img;

ComplexNumbers(**double** r, **double** i){

**this**.real = r;

**this**.img =i;

}

**public** **static** ComplexNumbers sum(ComplexNumbers c1, ComplexNumbers c2) {

ComplexNumbers temp1 = **new** ComplexNumbers(0, 0);

temp1.real = c1.real + c2.real;

temp1.img = c1.img + c2.img;

**return** temp1;

}

**public** **static** ComplexNumbers sub(ComplexNumbers c1, ComplexNumbers c2) {

ComplexNumbers temp2 = **new** ComplexNumbers(0, 0);

temp2.real = c1.real - c2.real;

temp2.img = c1.img - c2.img;

**return** temp2;

}

**public** **static** ComplexNumbers mul(ComplexNumbers c1, ComplexNumbers c2) {

ComplexNumbers temp3 = **new** ComplexNumbers(0, 0);

temp3.real = c1.real \* c2.real;

temp3.img = c1.img \* c2.img;

**return** temp3;

}

**public** **static** ComplexNumbers div(ComplexNumbers c1, ComplexNumbers c2) {

ComplexNumbers temp4 = **new** ComplexNumbers(0, 0);

temp4.real = c1.real / c2.real;

temp4.img = c1.img / c2.img;

**return** temp4;

}

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

// **TODO** Auto-generated method stub

ComplexNumbers c1 = **new** ComplexNumbers(4.5, 5);

ComplexNumbers c2 = **new** ComplexNumbers(1.5, 3.5);

ComplexNumbers temp1 = *sum*(c1, c2);

ComplexNumbers temp2 = *sub*(c1, c2);

ComplexNumbers temp3 = *mul*(c1, c2);

ComplexNumbers temp4 = *div*(c1, c2);

System.***out***.printf("Sum is: "+ temp1.real+" + "+ temp1.img +"i");

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

System.***out***.printf("Sub is: "+ temp2.real+" - "+ temp2.img +"i");

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

System.***out***.printf("Mul is: "+ temp3.real+" \* "+ temp3.img +"i");

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

System.***out***.printf("Div is: "+ temp4.real+" / "+ temp4.img +"i");

}

}

Output:

Sum is: 6.0 + 8.5i

Sub is: 3.0 - 1.5i

Mul is: 6.75 \* 17.5i

Div is: 3.0 / 1.4285714285714286i

**3.Write a class Account & add methods like deposit, withdraw, print etc.**

**package** com.zensar;

**public** **class** Bank {

**long** bal = 200000;

**long** minBalance = 5000;

**long** maxTransAmount = 60000;

**void** depsoitAmount(**int** amount) {

**if** (!(amount < maxTransAmount))

System.***out***.println("reduce your amount for deposit ");

bal += amount;

System.***out***.println("New Balance After Deposit: " + bal);

}

**void** withdrawAmount(**int** amount) {

**if** ((amount > maxTransAmount))

System.***out***.println("reduce your amount for withdraw");

**long** tempBal = bal - amount;

**if** ((tempBal < minBalance))

System.***out***.println("cannot withdraw");

bal = tempBal;

System.***out***.println("New Balance After withdrawal: " + bal);

}

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

Bank b = **new** Bank();

b.depsoitAmount(4000);

b.withdrawAmount(3000);

}

}

**Output:**

New Balance After Deposit: 204000

New Balance After withdrawal: 201000

**4.Write a program to implement a Stack using arrays as follows-**

**class StackedArray {**

**int ary[];**

**push(--) { }**

**pop() {--) {}**

**}**

**package** com.zensar;

**public** **class** StackArrayImple {

**int** size;

**int** arr[];

**int** top;

StackArrayImple(**int** size) {

**this**.size = size;

**this**.arr = **new** **int**[size];

**this**.top = -1;

}

// this method inserts an element on stack

**public** **void** push(**int** element) {

**if** (!isStackFull()) {

top++;

arr[top] = element;

System.***out***.println("Element Pushed on Stack is :" + element);

} **else** {

System.***out***.println ("Cannot insert Stack is full...");

}

}

// this method deletes an element from stack

**public** **int** pop() {

**if** (!isEmpty()) {

**int** returnedtopElement = top;

top--;

System.***out***.println("Element Popped from Stack is :" + arr[returnedtopElement]);

**return** arr[returnedtopElement];

} **else** {

System.***out***.println("Stack is empty...");

**return** -1;

}

}

// this method returns topmost element from stack

**public** **int** peek() {

**if**(!**this**.isEmpty())

**return** arr[top];

**else**

{

System.***out***.println("Stack is Empty");

**return** -1;

}

}

// this method checks stack is empty

**public** **boolean** isEmpty() {

**return** (top == -1);

}

**public** **boolean** isStackFull() {

**return** (size - 1 == top);

}

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

// **TODO** Auto-generated method stub

StackArrayImple impl = **new** StackArrayImple(10);

impl.pop();

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

impl.push(35);

impl.push(31);

impl.push(50);

impl.push(40);

impl.push(41);

impl.push(61);

impl.push(70);

impl.push(4);

impl.push(1);

impl.push(20);

impl.push(21);

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

impl.pop();

impl.pop();

impl.pop();

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

}

}

**Output:**

Stack is empty...

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

Element Pushed on Stack is :35

Element Pushed on Stack is :31

Element Pushed on Stack is :50

Element Pushed on Stack is :40

Element Pushed on Stack is :41

Element Pushed on Stack is :61

Element Pushed on Stack is :70

Element Pushed on Stack is :4

Element Pushed on Stack is :1

Element Pushed on Stack is :20

Cannot insert Stack is full...

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

Element Popped from Stack is :20

Element Popped from Stack is :1

Element Popped from Stack is :4

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

**5.Write a program to implement a Queue using arrays as follows-**

**class QueuedArray {**

**int ary[];**

**push(--) { }**

**pop() {--) {}**

**}**

**package** com.zensar;

**public** **class** QueueArrayImple {

**private** **int** capacity;

**int** queueArr[];

**int** front;

**int** rear;

**int** currentSize = 0;

**public** QueueArrayImple(**int** sizeOfQueue) {

**this**.capacity = sizeOfQueue;

front = 0;

rear = -1;

queueArr = **new** **int**[**this**.capacity];

}

//this method is used to add element in the queue

**public** **void** enqueue(**int** data) {

**if** (isFull()) {

System.***out***.println("Queue is full!! Can not add more elements");

} **else** {

rear++;

**if** (rear == capacity - 1) {

rear = 0;

}

queueArr[rear] = data;

currentSize++;

System.***out***.println(data + " added to the queue");

}

}

//This method removes an element from the front of the queue

**public** **void** dequeue() {

**if** (isEmpty()) {

System.***out***.println("Queue is empty!! Can not dequeue element");

} **else** {

front++;

**if** (front == capacity - 1) {

System.***out***.println(queueArr[front - 1] + " removed from the queue");

front = 0;

} **else** {

System.***out***.println(queueArr[front - 1] + " removed from the queue");

}

currentSize--;

}

}

//This method is use to check if element is full or not

**public** **boolean** isFull() {

**if** (currentSize == capacity) {

**return** **true**;

}

**return** **false**;

}

//This method is use to check if element is empty or not

**public** **boolean** isEmpty() {

**if** (currentSize == 0) {

**return** **true**;

}

**return** **false**;

}

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

// **TODO** Auto-generated method stub

QueueArrayImple queue = **new** QueueArrayImple(5);

queue.enqueue(6);

queue.dequeue();

queue.enqueue(3);

queue.enqueue(99);

queue.enqueue(56);

queue.dequeue();

queue.enqueue(43);

queue.dequeue();

queue.enqueue(89);

queue.enqueue(77);

queue.dequeue();

queue.enqueue(32);

queue.enqueue(232);

}

}

**Output:**

6 added to the queue

6 removed from the queue

3 added to the queue

99 added to the queue

56 added to the queue

3 removed from the queue

43 added to the queue

99 removed from the queue

89 added to the queue

77 added to the queue

56 removed from the queue

32 added to the queue

232 added to the queue

6**.Write a single tone class. Confirm that single tone class cannot be inherited.**

**package** com.zensar;

**public** **class** Singleton {

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

Example obj = Example.*getObject*();

}

}

**final** **class** Example {

**private** **static** Example *single\_obj* = **null**;

**public** String str;

**static** Example *obj* = **new** Example();

**private** Example() {

str = "Demo";

}

**public** **static** Example getObject() {

**if** (*single\_obj* == **null**)

*single\_obj* = **new** Example();

System.***out***.println("Singleton obj created");

**return** *single\_obj*;

}

}

**7.Write java classes to build doubly linked list. Add functionalities like add new node, insert node, delete node, count nodes & print linked list.**

**class Node {**

**Node previous;**

**Node next;**

**Int data;**

**}**

**package** com.zensar;

**class** DLLImpl {

Node head, tail = **null**;

**class** Node {

**int** data;

Node prev;

Node next;

Node(**int** d) {

data = d;

}

}

**void** insert(**int** data) {

Node new\_node = **new** Node(data);

**if** (head == **null**) {

head = tail = new\_node;

head.prev = **null**;

tail.next = **null**;

}

tail.next = new\_node;

new\_node.prev = tail;

tail = new\_node;

new\_node.next = **null**;

}

**void** delete(Node del) {

**if** (head == **null**) {

**return**;

}

**if** (head == del) {

head = del.next;

}

**if** (del.next != **null**) {

del.next.prev = del.prev;

}

**if** (del.prev != **null**) {

del.prev.next = del.next;

}

**return**;

}

**void** printNodes() {

Node curr = head;

**if** (head == **null**) {

System.***out***.println("DLL is empty");

**return**;

}

**while** (curr != **null**) {

System.***out***.print(curr.data + "->");

curr = curr.next;

}

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

}

}

**public** **class** DoublyLL {

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

DLLImpl dl = **new** DLLImpl();

dl.insert(10);

dl.insert(20);

dl.insert(30);

dl.insert(40);

dl.insert(50);

dl.printNodes();

dl.delete(dl.head.next);

dl.delete(dl.tail.prev);

System.***out***.println("after deletion");

dl.printNodes();

}

}

**Output:**

10->20->30->40->50->

after deletion

10->30->50->