DAY 5

/\*

Method Overriding: Redefining Parent class method in Child class.

Conditions:

1) Function signaturemust be same.

2) We cannot make it more stricter. ( We can keep the same level of strictness or lower)

Here, decision regarding which definition of accl() is to be invoked is done at runtime.

Binding takes place at runtime, so its called Dynamic Binding or Late Binding.

Hence, Runtime polymorphism is involved..

\*/

// Method Overriding

/\*

class Vehicle {

void accl() {

System.out.println("Accelerating at 100kmph");

}

void brake() {

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

}

}

class Car extends Vehicle {

void accl() {

System.out.println("Accelerating at 120kmph");

}

public static void main(String[] args) {

Car ob = new Car();

ob.accl();

ob.brake();

}

}

\*/

// Printng Object (Overriding toString())

/\*

import java.util.Scanner;

class Test {

int data;

public void input() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter Test data: ");

data = sc.nextInt();

}

public String toString() {

// return String.valueOf(data); //convert int to String in java

return Integer.toString(data);

}

}

class TestDrive {

public static void main(String[] args) {

Test ob = new Test();

ob.input();

System.out.println("Data: " + ob); //ob.toString()

}

}

\*/

/\*

class Sum {

int a, b, res;

Sum(int a, int b) {

this.a = a;

this.b = b;

}

void add() {

res = a + b;

}

public String toString() {

return "Sum = " + res;

}

}

class Result {

public static void main(String[] args) {

Sum s = new Sum(25, 38);

s.add();

System.out.println(s); //s.toString()

}

}

\*/

/\*

Abstraction can be implemented using:

a) Abstract class (> 0% & <= 100%)

b) Interface (100% Abstraction)

Abstract Class: A class which can't be instantiated. (Cannot create object)

If a class contains an "abstract method" then the class must be abstract class.

Abstract Method: A method which has no definition. (No function body)

Note: If a class is abstract, then it is not necessary that all its methods would be abstract.

Abstract Method will contain both abstract method(s) and concrete/complete method(s).

=> We must Implement an Abstract Class.

=> A child class of the Abstract Class can implement it.

=> The child class must define all the abstract methods of parent abstract class.

\*/

/\*

// Implementation of Abstract Class

//Level - 1 User

abstract class Music {

//Abstract method

abstract void play();

//Complete method

void bass() {

System.out.println("Extra Bass...");

}

}

//Level - 2 User

class MusicDevice extends Music {

void play() {

System.out.println("Playing Music...");

}

}

//Level - 3 User

class Player {

public static void main(String[] args) {

MusicDevice md = new MusicDevice();

md.play();

md.bass();

}

}

\*/

/\*

// Implementation of Interface:

// All methods are abstract & public by default

// All data members are final, static and public by default

// We need to create an implementing class to drive Interface

interface Music {

void play(); //abstract public void play();

void bass();

}

class MusicDevice implements Music {

public void play() {

System.out.println("Playing Music...");

}

public void bass() {

System.out.println("Extra Bass...");

}

}

class Player {

public static void main(String[] args) {

MusicDevice md = new MusicDevice();

md.play();

md.bass();

}

}

\*/

/\*

Multiple Inheritance using class is not supported but by using Interface is possible

in Java.

\*/

/\*

interface Drawable {

void paint();

}

interface Sketchable {

void paint();

}

interface Printable extends Drawable, Sketchable {

}

class Print implements Printable {

public void paint() {

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

}

}

class Driver {

public static void main(String[] args) {

Print p = new Print();

p.paint();

}

}

\*/

/\*

import java.util.Scanner;

class Test {

static void display(int a[]) {

System.out.print("Array elements: ");

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

System.out.print(a[i] + " ");

}

}

public static void main(String[] args) {

int arr[], n;

Scanner sc = new Scanner(System.in);

System.out.print("Enter the array size: ");

n = sc.nextInt();

arr = new int[n];

System.out.print("Enter array elements: ");

for(int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

display(arr);

}

}

\*/

//For each loop

import java.util.Scanner;

class Test {

static void display(int a[]) {

System.out.print("Array elements: ");

for(int i : a) {

System.out.print(i + " ");

}

}

public static void main(String[] args) {

int arr[], n;

Scanner sc = new Scanner(System.in);

System.out.print("Enter the array size: ");

n = sc.nextInt();

arr = new int[n];

System.out.print("Enter array elements: ");

for(int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

display(arr);

}

}