**ABDUL BASIT**

**193227**

**BSCS-6C**

**TASK#1**

/\*\*

\*

\* @author abasit.bscs16seecs

\*/

public class Person implements PersonInterface,AnotherPersonInterface {

int cashSaving;

int retirementFund;

String firstName;

String lastName;

public Person(int c,int r,String f,String l) {

cashSaving=c;

retirementFund=r;

firstName=f;

lastName=l;

}

@Override

public int computeTotalWealth(){

return cashSaving+retirementFund;

}

public String getName(){

return firstName+lastName;

}

@Override

public int measureIntelligence(String s){

return (cashSaving\*10/retirementFund)\*10;

}

}

public interface PersonInterface {

int computeTotalWealth();

String getName();

}

public interface AnotherPersonInterface {

int measureIntelligence(String s);

}

public class Test {

public static void main(String[] args) {

// Create an object instance of Person class.

Person person1 = new Person(10000, 20000, "Abdul", " Basit");

// You can assign the object instance to

// PersonInterface type.

PersonInterface personinterface1 = person1;

// Display data from person1 and personinterface1.

// Observe that they refer to the same object instance.

System.out.println( "person1.getName() = " + person1.getName() +

"," + " person1.computeTotalWealth() = " +

person1.computeTotalWealth() + ",\n" +

"person1.measureIntelligence() = " +

person1.measureIntelligence(person1.getName()));

System.out.println( "personinterface1.getName() = " +

personinterface1.getName() + ",\n" +

"personinterface1.computeTotalWealth() = " +

personinterface1.computeTotalWealth());

// You can assign the object instance to

// AnotherPersonInterface type.

AnotherPersonInterface anotherpersoninterface1 = person1;

// Check of object instance that is referred by personinterface1 and

// anotherpersoninterface1 is the same object instance.

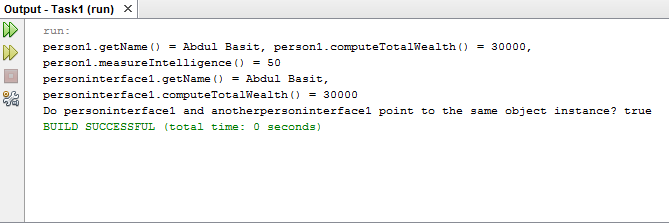
boolean b1 = (personinterface1 == anotherpersoninterface1);

System.out.println("Do personinterface1 and anotherpersoninterface1 point to the same object instance? " + b1);

}

}

**OUTPUT**



**TASK#2**

/\*\*

\*

\* @author Abdul Basit

\*/

public class Line implements RelationInterface{

double x1;

double x2;

double y1;

double y2;

public Line(double x1,double x2,double y1,double y2) {

this.x1=x1;

this.x2=x2;

this.y1=y1;

this.y2=y2;

}

public double getLength(){

return Math.pow((Math.pow((x2-x1),2)+Math.pow((y2-y1),2)),0.5);

}

public boolean isGreater(Line a,Line b){

if(a.getLength()>b.getLength())

return true;

else

return false;

}

public boolean isLess(Line a,Line b){

if(a.getLength()<b.getLength())

return true;

else

return false;

}

public boolean isEqual(Line a,Line b){

if(a.getLength()==b.getLength())

return true;

else

return false;

}

}

public interface RelationInterface {

boolean isGreater(Line a,Line b);

boolean isLess(Line a,Line b);

boolean isEqual(Line a,Line b);

}

public class Test {

public static void main(String[] args) {

// TODO code application logic here

// Create two Line object instances.

Line line1 = new Line(1.0, 2.0, 1.0, 2.0);

Line line2 = new Line(2.0, 3.0, 2.0, 3.0);

boolean b1 = line1.isGreater(line1, line2);

System.out.println("line1 is greater than line2: " + b1);

boolean b2 = line1.isEqual(line1, line2);

System.out.println("line1 is equal with line2: " + b2);

// Note that the line3 is object instance of Line type.

// Because the Line type is also a type of RelationInterface,

// the line3 variable can be declared as RelationInterface type.

// This is a very very important concept you need to understand.

RelationInterface line3 = new Line(1.0, 5.0, 1.0, 5.0);

boolean b3 = line3.isEqual(line1, (Line)line3);

System.out.println("line1 is equal with line3: " + b3);

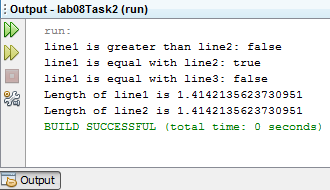
System.out.println("Length of line1 is " + line1.getLength());

System.out.println("Length of line2 is " + line2.getLength());

}

}

**OUTPUT**

****

**TASK#3**

public interface ProductInterface {

double computeSalePrice();

double getRegularPrice();

void setRegularPrice(double a);

}

public interface ElectronicsInterface {

String getManufacturer();

}

public class Product implements ProductInterface{

private double regularPrice;

public Product(double a) {

regularPrice=a;

}

public double getRegularPrice() {

return regularPrice;

}

public void setRegularPrice(double regularPrice) {

this.regularPrice = regularPrice;

}

public double computeSalePrice(){

return getRegularPrice()\*0.8;

}

}

public class Electronics extends Product implements ElectronicsInterface{

private String manufacturer;

public Electronics(double a,String manufacturer) {

super(a);

this.manufacturer = manufacturer;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

public class Book extends Product{

private String pulisher;

private int yearPublished;

public Book( double a,String pulisher, int yearPublished) {

super(a);

this.pulisher = pulisher;

this.yearPublished = yearPublished;

}

public String getPulisher() {

return pulisher;

}

public void setPulisher(String pulisher) {

this.pulisher = pulisher;

}

public int getYearPublished() {

return yearPublished;

}

public void setYearPublished(int yearPublished) {

this.yearPublished = yearPublished;

}

}

public class TV extends Electronics {

private int size;

public TV(double a, String manufacturer,int size) {

super(a, manufacturer);

this.size = size;

}

@Override

public double computeSalePrice() {

return super.computeSalePrice(); //To change body of generated methods, choose Tools | Templates.

}

}

public class MP3Player extends Electronics{

private String color;

public MP3Player(double a, String manufacturer,String color) {

super(a, manufacturer);

this.color = color;

}

@Override

public double computeSalePrice() {

return super.computeSalePrice(); //To change body of generated methods, choose Tools | Templates.

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

}

public class Test {

public static void main(String[] args) {

// TODO code application logic here

// Declare and create Product array of size 5

ProductInterface[] pa = new ProductInterface[5];

// Create object instances and assign them to the type of Product.

pa[0] = new TV(1000, "Samsung", 30);

pa[1] = new TV(2000, "Sony", 50);

pa[2] = new MP3Player(250, "Apple", "blue");

pa[3] = new Book(34, "Sun press", 1992);

pa[4] = new Book(15, "Korea press", 1986);

// Compute total regular price and total sale price.

double totalRegularPrice = 0;

double totalSalePrice = 0;

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

{

// Call a method of the super class to get the regular price.

totalRegularPrice += pa[i].getRegularPrice();

// Since the sale price is computed differently

// depending on the product type, overriding (implementation)

// method of the object instance of the sub-class

// gets invoked. This is runtime polymorphic behavior.

totalSalePrice += pa[i].computeSalePrice();

System.out.println("Item number " + i + ": Type = " + pa[i].getClass().getName() +

", Regular price = " + pa[i].getRegularPrice() +

", Sale price = " + pa[i].computeSalePrice());

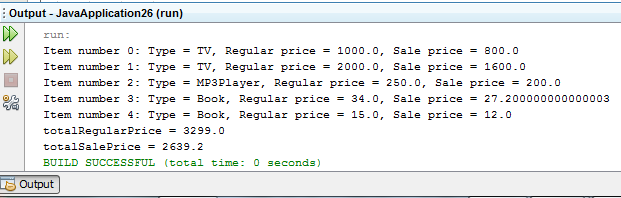
}

System.out.println("totalRegularPrice = " + totalRegularPrice);

System.out.println("totalSalePrice = " + totalSalePrice);

}

}

**OUTPUT**