**Muhammad Rizwan Khalid**

**180459**

**BSCS 6A**

**Lab no: 08**

**Task one:**

package lab08;

public class Lab08 {

public static void main(String[] args) {

// Create an object instance of Person class.

Person person1 = new Person(10000, 20000, "Quintin", "Tarantino");

// 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() + "," +

" person1.measureIntelligence() = " +

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

System.out.println(

"personinterface1.getName() = " + personinterface1.getName() + "," +

" 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);

}

}

interface PersonInterface{

int computeTotalWealth();

String getName();

}

interface AnotherPersonInterface{

int measureIntelligence(String s);

}

class Person implements PersonInterface,AnotherPersonInterface{

int cashSaving;

int retirementFund;

String firstName;

String lastName;

public Person(int cashSaving, int retirementFund, String firstName, String lastName) {

this.cashSaving = cashSaving;

this.retirementFund = retirementFund;

this.firstName = firstName;

this.lastName = lastName;

}

@Override

public int computeTotalWealth(){

return cashSaving + retirementFund;

}

@Override

public String getName(){

return String.format(firstName + " " + lastName);

}

@Override

public int measureIntelligence(String s){

int sum = 0;

for(int i = 0; i<s.length()/2;i++){

sum = sum + s.charAt(i);

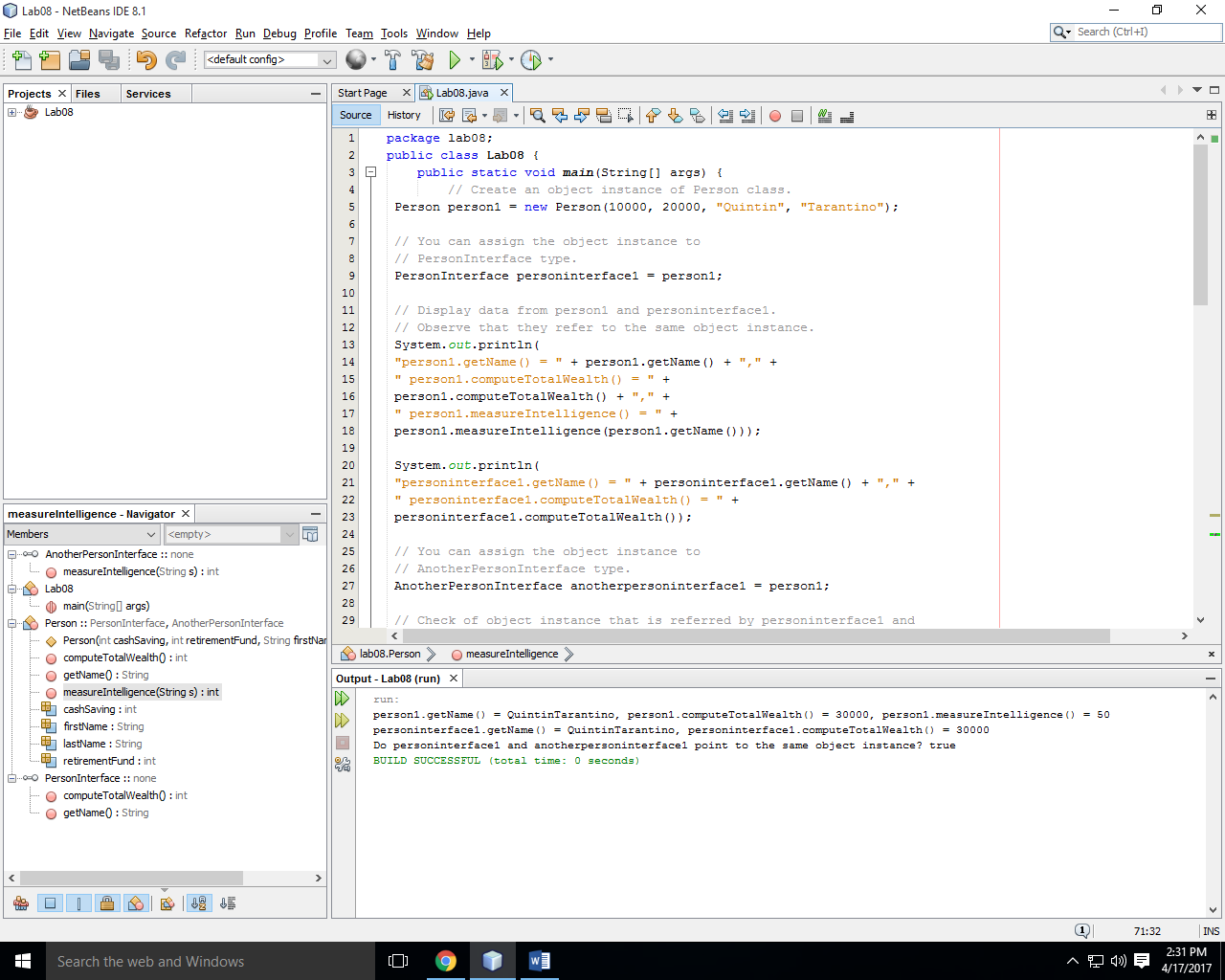
}

return sum/s.length() + 5;

}

}

**Output:**



**Question:** Compile error is expected on the following line of code. What could be causing it? personinterface1.measureIntelligence(personinterface1.getName());

**Answer:** There is an error because the signature measureIntelligence is present in interface AnotherPersonInterface. If interface PersonInterface extends the AnotherPersonInterface then the error is eliminated;

**Task two:**

package lab08;

public class Lab08 {

public static void main(String[] args) {

// 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());

}

}

interface RelationInterface{

boolean isGreater(Line a, Line b);

boolean isLess(Line a, Line b);

boolean isEqual(Line a, Line b);

}

class Line implements RelationInterface{

private double x1;

private double x2;

private double y1;

private 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(){

double a = Math.pow(x2-x1,2) + Math.pow(y2-y1,2);

return Math.pow(a,0.5);

}

@Override

public boolean isGreater(Line a, Line b){

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

return true;

}

else

return false;

}

@Override

public boolean isLess(Line a, Line b){

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

return true;

}

else

return false;

}

@Override

public boolean isEqual(Line a, Line b){

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

return true;

}

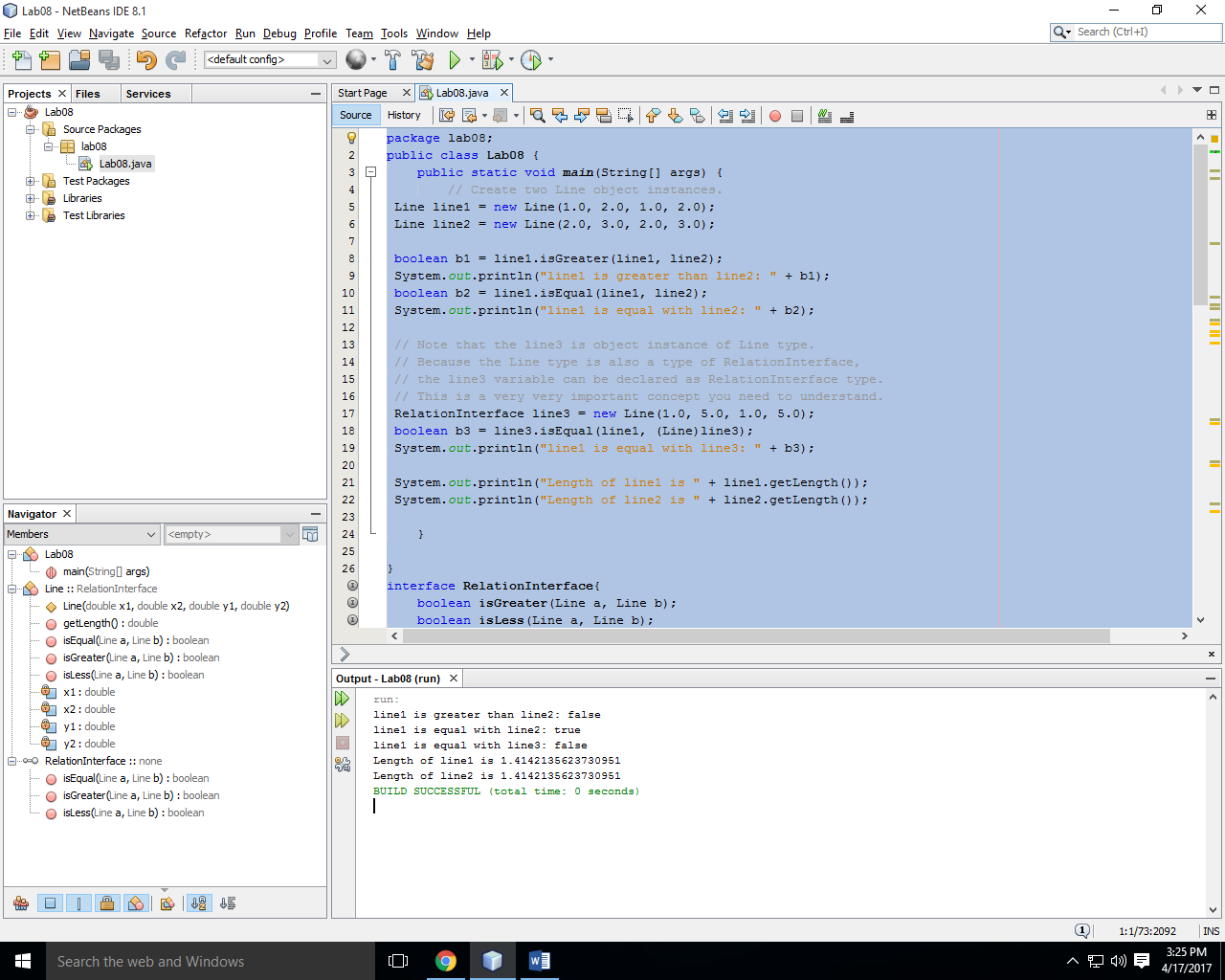
else

return false;

}

}

**Output:**



**Task three:**

package lab08;

public class Lab08 {

public static void main(String[] args) {

// Declare and create Product array of size 5

ProductInterface[] pa = new Product[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);

}

}

interface ProductInterface{

double computeSalePrice();

double getRegularPrice();

void setRegularPrice(double regularPrice);

}

interface ElectronicsInterface{

String getManufacturer();

}

class Product implements ProductInterface{

private double regularPrice;

public Product(double regularPrice) {

this.regularPrice = regularPrice;

}

@Override

public void setRegularPrice(double regularPrice){

this.regularPrice = regularPrice;

}

@Override

public double getRegularPrice(){

return this.regularPrice;

}

@Override

public double computeSalePrice(){

return 0;

}

}

class Book extends Product{

private String publisher;

private int yearPublished;

public Book(double regularPrice, String publisher, int yearPublished) {

super(regularPrice);

this.publisher = publisher;

this.yearPublished = yearPublished;

}

public void setPublisher(String publisher){

this.publisher = publisher;

}

public void setYearPublished(int yearPublished){

this.yearPublished = yearPublished;

}

public String getPublisher(){

return this.publisher;

}

public int yearPublished(){

return this.yearPublished;

}

@Override

public double computeSalePrice(){

return 0.5 \* getRegularPrice();

}

}

class Electronics extends Product implements ElectronicsInterface{

private String manufacturer;

public Electronics(double regularPrice, String manufacturer) {

super(regularPrice);

this.manufacturer = manufacturer;

}

public void setManufacturer(String manufacturer){

this.manufacturer = manufacturer;

}

@Override

public String getManufacturer(){

return this.manufacturer;

}

}

class MP3Player extends Electronics {

private String color;

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

super(regularPrice,manufacturer);

this.color = color;

}

public void setColor(String color){

this.color = color;

}

public String getColor(){

return this.color;

}

@Override

public double computeSalePrice(){

return 0.9 \* getRegularPrice();

}

}

class TV extends Electronics{

private int size;

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

super(regularPrice,manufacturer);

this.size = size;

}

@Override

public double computeSalePrice(){

return 0.8 \* getRegularPrice();

}

}

**Output:**

