**ABDUL BASIT**

**193227**

**BSCS-6C**

**TASK#1**

**CODE**

public abstract class LivingThing {//CREATING ABSTRACT CLASS LIVINGTHING

private String name;//FILEDS

public LivingThing(String n) {//CONSTRUCTOR

name=n;}

public void breath(){}//FUNCTIONS

public void eat(){}

public abstract void walk();//ABSTRACT FUNCTION

//SETTER AND GETTERS

public String getName(){

return name;}

public void setName(String n){

name=n;}

}

//HUMAN CLASS EXTENDS LIVINGTHING

public class Human extends LivingThing{

public Human(String n) {

super(n);

}

public void walk(){//OVERRIDDING

System.out.printf("Human Will Rodman walks...\n");}

}

//CREATING MONKEY CLASS

public class Monkey extends LivingThing{

public Monkey(String n) {

super(n);

}

public void walk(){//OVERRIDDING

System.out.printf("Monkey Caesar also walks...\n");}

}

//MAIN STARTS

public class Main {

public static void main( String[] args) {

// Create Human object instance

// and assign it to Human type.

Human human1 = new Human( "Will Rodman");

human1.walk();

// Create Human object instance

// and assign it to LivingThing type.

LivingThing livingthing1 = human1;

livingthing1.walk();

// Create a Monkey object instance

// and assign it to LivingThing type.

LivingThing livingthing2 = new Monkey( "Caesar");

livingthing2.walk();

// Display data from human1 and livingthing1.

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

System.out.println( "human1.getName() = " + human1.getName());

System.out.println( "livingthing1.getName() = " +

livingthing1.getName());

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

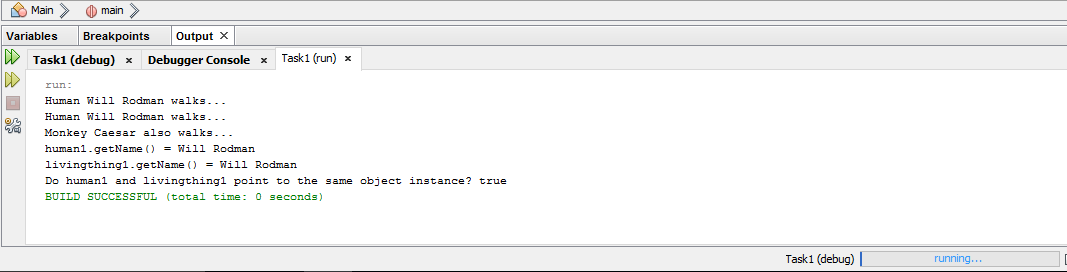
// y is the same object instance.

boolean b1 = ( human1 == livingthing1);

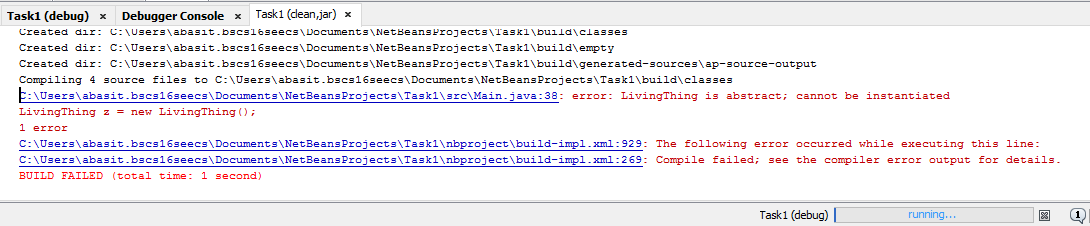
System.out.println( "Do human1 and livingthing1 point to the same object instance? " + b1);

}

}**OUTPUT**



**BONUS**

The error occurs because the class of which objects we are creating is abstract and we cannot creates its objects, but we can use polymorphism to use the reference variables of the abstract class.

**TASK#2**

/\*\*

\*

\* @author abasit.bscs16seecs

\*/

public abstract class Product {//CREATING PRODUCT CLASS

double regularPrice;//FIELDS

public Product(double rprice) {//CONSTRUCTOR

regularPrice=rprice;

}

//ABSTRACT METHOS WHICH TO WE HAVE TO OVERRIDDEN

public abstract double computeSalePrice();

public double getRegularPrice(){

return regularPrice;

}

public void setRegularPrice(double rp){

regularPrice=rp;

}

}

//ELECTRONICS CLASS EXTENDING PRODUCT CLASS

public abstract class Electronics extends Product {

private String manufacturer;//FILEDS

public Electronics(double rprice,String m) {

super(rprice);//CALLING SUPER CALL CONSTRUCTOR

setManufacturer(m);//CALLING SETTER FUNCTION

}

//GETTER AND SETTERS

public String getManufacturer(){

return manufacturer;

}

public void setManufacturer(String m){

manufacturer=m;

}

}

public class Book extends Product {

public String publisher;

public int yearPublished;

public Book(double rprice,String p,int y) {

super(rprice);

setPublisher(p);

setYearPublished(y);

}

@Override

public double computeSalePrice(){

return getRegularPrice()/2;

}

public String getPublisher(){

return publisher;

}

public void setPublisher(String p){

publisher=p;

}

public int getYearPublisehed(){

return yearPublished;

}

public void setYearPublished(int yp){

yearPublished=yp;

}

}

public class MP3Player extends Electronics {

String color;

public MP3Player(double rprice, String m,String c) {

super(rprice, m);

setColor(c);

}

@Override

public double computeSalePrice(){

return getRegularPrice()-getRegularPrice()\*0.1;

}

public void setColor(String c){

color=c;

}

public String getColor(){

return color;

}

}

public class TV extends Electronics{

int size;

public TV(double rprice, String m,int s) {

super(rprice, m);

setSize(s);

}

@Override

public double computeSalePrice(){

return getRegularPrice()-getRegularPrice()\*0.2;

}

public void setSize(int s){

size=s;

}

public int getSize(){

return size;

}

}

public class Main {

public static void main(String[] args) {

// Declare and create Product array of size 5

Product[] 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);

}}

**OUTPUT**

