Name：إياد امين احمد احمد الفار

Student ID：202000172

Sheet: 3

[Due Date]

**9.1**

(*The* ***Rectangle*** *class*) Following the example of the **Circle** class in Section 9.2,

design a class named **Rectangle** to represent a rectangle.

Sol:

public class Test{

    public static void main(String[] args) {

        rectangle r1= new rectangle(40,4);

        System.out.println(" r1 width = " + r1.width + " r1 height = " +

        r1.height + " r1 Area = " + r1.getArea() + " r1 perimeter = " + r1.getPerimeter() );

        rectangle r2= new rectangle(3.5,35.9);

        System.out.println(" r2 width = " + r2.width + " r2 height = " +

        r2.height + " r2 Area = " + r2.getArea() + " r2 perimeter = " + r2.getPerimeter() );

    }}

class rectangle {

    double height = 1;

    double width = 1 ;

    rectangle(){

    }

    rectangle(double h,double w){

        height = h;

        width = w;

    }

    public double getArea(){

        return height\*width;

    }

    public double getPerimeter(){

        return 2\*(height+width);

    }}

**9.2**

(*The* ***Stock*** *class*) Following the example of the **Circle** class in Section 9.2,

design a class named **Stock**

Sol:

public class Test{

    public static void main(String[] args) {

        Stock s1= new Stock("Oracle Corporation","ORCL");

        s1.previousClosingPrice =34.5;

        s1.currentPrice =34.35;

        System.out.println(" name is: " + s1.name + " symbol is: " +

        s1.symbol + " Price change percent is: " + s1.getChangePercent() + "%");

    }}

class Stock {

    String symbol = new String();

    String name = new String();

    double previousClosingPrice = 1;

    double currentPrice = 1 ;

    Stock(String n,String sym){

        symbol = sym;

        name = n;

    }

    public double getChangePercent(){

        double x = Math.abs(((currentPrice - previousClosingPrice)/previousClosingPrice) \* 100);

        x \*= 1000;

        x = Math.round(x);

        x /=1000;

        return x;

    }}

**\*9.3**

(*Use the* ***Date*** *class*) Write a program that creates a **Date** object, sets its elapsed

time to **10000**, **100000**, **1000000**, **10000000**, **100000000**, **1000000000**,

**10000000000**, and **100000000000**, and displays the date and time using the

**toString()** method, respectively.

Sol:

import java.util.Date;

public class Test{

    public static void main(String[] args) {

        Date[] dates = new Date[8];

        int sum = 10000;

        for (int i = 0;i<8;i++){

            dates[i] = new Date(sum);

            sum \*= 10;

        }

        for (int i = 0;i<8;i++){

            System.out.println(" date "+ (i+1) +" is: " + dates[i].toString());

        }}}

**\*9.4**

(*Use the* ***Random*** *class*) Write a program that creates a **Random** object with seed

**1000** and displays the first 50 random integers between **0** and **100** using the

**nextInt(100)** method.

Sol:

import java.util.Random;

public class Test{

    public static void main(String[] args) {

        Random r1 = new Random(1000);

        for (int i = 0;i<50;i++){

           System.out.println(r1.nextInt(100));

        }}}

**\*9.5**

(*Use the* ***GregorianCalendar*** *class*).

Write a program to perform two tasks:

1. Display the current year, month, and day.

2. The **GregorianCalendar** class has the **setTimeInMillis(long)**, which can be used to set a specified elapsed time since January 1, 1970. Set the value to **1234567898765L** and display the year, month, and day.

Sol:

import java.util.GregorianCalendar;

import java.util.Date;

public class Test{

public static void main(String[] args) {

    Date d1 = new Date();

    System.out.println(d1.toString());

    GregorianCalendar gc1 = new GregorianCalendar();

    gc1.setTimeInMillis(1234567898765L);

    System.out.println("Year: " + gc1.get(GregorianCalendar.YEAR) +

    " Month: " + gc1.get(GregorianCalendar.MONTH) +

    " Day: " + gc1.get(GregorianCalendar.DATE));

}}

**\*9.6**

(*Stopwatch*) Design a class named **StopWatch**. The class contains:

■ Private data fields **startTime** and **endTime** with getter methods.

■ A no-arg constructor that initializes **startTime** with the current time.

■A method named **start()** that resets the **startTime** to the current time.

■A method named **stop()** that sets the **endTime** to the current time.

■ A method named **getElapsedTime()** that returns the elapsed time for the stopwatch in milliseconds.

Sol:

import java.util.concurrent.TimeUnit;

public class Test{

public static void main(String[] args) {

    Stopwatch sw1= new Stopwatch();

    long[] arr = new long[100000];

    for (int i = 0;i < arr.length;i++){arr[i] = (long)Math.random();}

    sw1.start();

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

        for (int j = i + 1; j < arr.length; j++) {

            long temp = 0;

            if (arr[j] < arr[i]) {

                temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;}}}

    sw1.stop();

    System.out.println("Elapsed time is :" + sw1.getElapsedTime());}}

class Stopwatch{

    private long startTime,endTime;

    Stopwatch(){startTime = System.currentTimeMillis();}

    public void start(){startTime = System.currentTimeMillis();}

    public void stop(){endTime = System.currentTimeMillis();}

    public long getElapsedTime(){return (endTime-startTime);}}

**9.8**

(*The* ***Fan*** *class*) Design a class named **Fan** to represent a fan. The class contains:

■Three constants named **SLOW**, **MEDIUM**, and **FAST** with the values **1**, **2**, and **3** to denote the fan speed.

■ A private **int** data field named **speed** that specifies the speed of the fan (the default is **SLOW**).

■ A private **boolean** data field named **on** that specifies whether the fan is on (the default is **false**).

■ A private **double** data field named **radius** that specifies the radius of the fan (the default is **5**).

■ A string data field named **color** that specifies the color of the fan (the default is **blue**).

■ The accessor and mutator methods for all four data fields.

■ A no-arg constructor that creates a default fan.

■ A method named **toString()** that returns a string description for the fan. If the fan is on, the method returns

the fan speed, color, and radius in one combined string. If the fan is not on,

the method returns the fan color and radius along with the string “fan is off” in one combined string.

**Write a test program that creates two Fan objects. Assign maximum speed, radius 10, color yellow,**

**and turn it on to the first object. Assign medium speed, radius 5, color blue, and turn it off to the second object.**

**Display the objects by invoking their toString method.**

Sol:

public class Test{

public static void main(String[] args){

    Fan f1=new Fan(),f2= new Fan();

    f1.turnOn();

    f1.setSpeed(Fan.FAST);

    f1.setRadius(10);

    f1.setColor("yellow");

    f2.setSpeed(Fan.MEDIUM);

  // fan defualt is off , radius defualt is 5 ,color defualt is blue

    f1.fanToString();

    f2.fanToString(); }}

class Fan {

    public static final int SLOW=1,MEDIUM=2,FAST=3;

    private int speed = SLOW;

    private boolean on = false;

    private double radius = 5;

    String color = "blue";

    Fan(){}

    public int setSpeed(int s){

        if(speed>0 && speed<4) {speed = s;return 0;} else return -1;}

public String getSpeed() {returnspeed==1?"SLOW(MIN)":speed==2?"MEDIUM":"FAST(MAX)";}

    public void incSpeed(){if(speed<FAST)speed++;}

    public void decSpeed(){if(speed>SLOW)speed--; }

    public void turnOn(){on = true;}

    public void turnOff(){on = false;}

    public boolean isOpen(){return on;}

    public void setColor(String col){color = col;}

    public String getColor(){return color;}

    public void setRadius(double rad){radius = rad;}

    public double getRadius(){return radius;}

    public void fanToString(){

        if(on == true){System.out.println("Fan details: Speed is : "+getSpeed()+", Fan color is : "+getColor()+", Fan radius is :" +getRadius());}

        else{System.out.println( "Fan is off , Fan color is : " + getColor() + ", Fan radius is : " + getRadius()); }}}

**\*9.10**

(*Algebra: quadratic equations*) Design a class named **QuadraticEquation** for

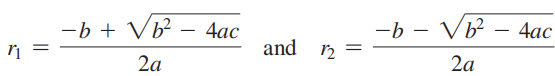
a quadratic equation *ax2* + *bx* + *c* = 0. The class contains:

■ Private data fields **a**, **b**, and **c** that represent three coefficients.

■ A constructor with the arguments for **a**, **b**, and **c**.

■ Three getter methods for **a**, **b**, and **c**.

■ A method named **getDiscriminant()** that returns the discriminant, which is *b2* -4*ac*.

■ The methods named **getRoot1()** and **getRoot2()**for returning two roots of the equation

These methods are useful only if the discriminant is non-negative.

Let these methods return **0** if the discriminant is negative.

Write a test program that prompts the user to enter values for *a*, *b*, and *c* and displays the result

based on the discriminant. If the discriminant is positive, display the two roots.

If the discriminant is 0, display the one root. Otherwise, display “The equation has no roots.”

Sol:

import java.util.Scanner;

public class Test{

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

double x,y,z;

System.out.println("Enter the value of the coefficients: ");

System.out.print("a = ");

x = input.nextDouble();

System.out.print("b = ");

y = input.nextDouble();

System.out.print("c = ");

z = input.nextDouble();

QuadraticEquation Qe1 = new QuadraticEquation(x, y, z);

System.out.println(Qe1.getDiscriminant());

if(Qe1.getDiscriminant()>0){

System.out.println("root 1 = " + Qe1.getRoot1());

System.out.println("root 2 = " + Qe1.getRoot2());}

else if(Qe1.getDiscriminant()==0){

System.out.println("the value of roots = " + Qe1.getRoot1());}

else

System.out.println("the equation has no roots");}}

class QuadraticEquation {

    private double a,b,c = 0;

    QuadraticEquation(double aa,double bb,double cc){

        a=aa;

        b=bb;

        c=cc; }

    public double get\_a(){return a;}

    public double get\_b(){return b;}

    public double get\_c(){return c;}

    public double getDiscriminant(){return (b\*b)-(4\*a\*c);}

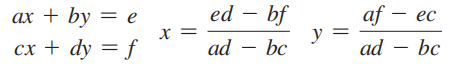
    public double getRoot1(){return getDiscriminant()>0?((-b)+Math.sqrt(getDiscriminant()))/(2\*a):0; }

    public double getRoot2(){return getDiscriminant()>0?((-b)-Math.sqrt(getDiscriminant()))/(2\*a):0;}}

**\*9.11**

(*Algebra:* 2 \* 2 *linear equations*) Design a class named **LinearEquation** for a

2 \* 2 system of linear equations:

The class contains:

■ Private data fields **a**, **b**, **c**, **d**, **e**, and **f**.

■ A constructor with the arguments for **a**, **b**, **c**, **d**, **e**, and **f**.

■ Six getter methods for **a**, **b**, **c**, **d**, **e**, and **f**.

■ A method named **isSolvable()** that returns true if *ad* - *bc* is not 0.

■ Methods **getX()** and **getY()** that return the solution for the equation.

Write a test program that prompts the user to enter **a**, **b**, **c**, **d**, **e**, and **f** and displays the result.

If *ad* - *bc* is 0, report that “The equation has no solution.”

Sol:

import java.util.Scanner;

public class Test{

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

double v[] = new double[6];

System.out.println("Enter the value of the coefficients: ");

LinearEquation.howTo();

for(int i=0;i<6;i++){

    System.out.print("#"+ (i+1) +" = ");

    v[i] = input.nextDouble();}

LinearEquation Le1 = new LinearEquation(v[0],v[1],v[2],v[3],v[4],v[5]);

if(Le1.isSolvable()){

    System.out.println("x = " + Le1.getX());

    System.out.println("y = " + Le1.getY()); }

else

    System.out.println("The equation has no solution.");}}

class LinearEquation {

    private double a,b,c,d,e,f = 0;

    LinearEquation(double aa,double bb,double cc,double dd,double ee,double ff){

        a=aa; b=bb; c=cc;

        d=dd; e=ee; f=ff; }

    public double get\_a(){return a;}

    public double get\_b(){return b;}

    public double get\_c(){return c;}

    public double get\_d(){return d;}

    public double get\_e(){return e;}

    public double get\_f(){return f;}

    public boolean isSolvable(){return (a\*d)-(b\*c)!=0?true:false;}

    public static void howTo(){System.out.println("ax+by=e , cx+dy=f");}

    public double getX(){

        double x=0;

        x= ((e\*d)-(b\*f))/((a\*d)-(b\*c)); return x; }

    public double getY(){

        double y=0;

        y= ((a\*f)-(e\*c))/((a\*d)-(b\*c)); return y;} }

Mini Project: The Account Class:

import java.util.Date;

public class Test{

public static void main(String[] args) {

    Account acc1 = new Account(1122,20000);

    Account.annualInterestRate = 4.5;

    acc1.withdraw(2500);

    acc1.deposit(3000);

    System.out.println("balance: "+acc1.getBalance() + " monthly interest rate: " +

    acc1.getMonthlyInterestRate()+ "%" + " date created: " + acc1.getDate() );

}}

class Account  {

    private int id = 0;

    private double balance = 0;

    static double annualInterestRate = 0;

    private Date dateCreated = new Date();

    Account(){}

    Account(int id,double balance){

        this.id = id;

        this.balance=balance;

    }

public int getID(){

return id;}

public double getBalance(){

return balance;}

public double getAnnualInterestRate(){

return annualInterestRate;}

public void setID(int id){

this.id = id;

}

public void setBalance(double balance){

this.balance= balance;

}

public void setAnnualInterestRate(double annualInterestRate){

this.annualInterestRate = annualInterestRate;

}

public double getMonthlyInterestRate(){

return annualInterestRate/12;}

public void withdraw(int w){

balance -= w;

}

public void deposit(int w){

balance += w;}

public Date getDate(){

return dateCreated;

}

}