JAVA BASIC

Lab Guides

Lab Specifications:

Create a class to represent a **Circle** type in java, which should have following:

- **radius**: an instance field of type double
- **No-argument constructor:** set radius with default value of 1.0
- **Constructor:** that accept an argument for radius
- **getRadius:** public method that returns the radius of Circle
- **getArea.** Method that returns the area of Circle.

Create a new class named Test with a main() method.

In main, creates a Circle object that initialized to a radius value of 12.

Questions:

- Explain the meaning of line 19 in class Circle: Math.***PI***, Math.*pow*(radius, 2).

Guidelines:

- Step1. Open Eclipse IDE
- Step2. Create a new project named **Exercise0302**
- Step3: Create two classes in turn named **Circle** and **Test** in this project

Circle class source code:

```
1.  package exercise;
2.  public class Circle {
3.      private double radius;
4.      public Circle() {
5.          this.radius = 1.0;
6.      }
7.      public Circle(double radius) {
8.          super();
9.          this.radius = radius;
10.     }
11.     public double getRadius() {
12.         return radius;
13.     }
14.     /**
15.     Calculate circle area based on radius.
16.     @return the area value of circle.
17.     */
18.     public double getArea() {
19.         return Math.PI * Math.pow(radius, 2);
```

```
20.     }
21. }
```

Test class source code:

```
1.  package exercise;
2.  public class Test {
3.      public static void main(String[] args) {
4.      // Create an instance of Circle
5.      Circle circle = new Circle(12);
6.      // Calls method
7.      System.out.println(String.format("Area of the circle %.2f", circle.getArea()));
8.      }
9.  }
```

» Step4: How to run

In Eclipse window I select **Run Test** or right-click **Run as..**:

Outputs

```
Area of the circle 452.39
```

Lab Specifications:

Write a Java class **Complex** for dealing with complex number. Your class must have the following features:

Instance variables:

- **realPart** for the real part of type double
- **imaginaryPart** for imaginary part of type double.

Constructor:

- public **Complex**(): A default constructor, it should initialize the number to 0, 0)
- public **Complex(double realPart, double imaginaryPart):** A constructor with parameters, it creates the complex object by setting the two fields to the passed values.

Instance methods:

- Getter/Setter Methods: are used to get/set the value.
- **public Complex add(Complex otherNumber)**: This method will find the sum of the current complex number and the passed complex number. The methods returns a new Complex number which is the sum of the two.
- **public String toString()**: This method allows the complex number to be easily printed out to the screen
- Write a separate class **ComplexDemo** with a main() method and test the Complex class methods.

Questions:

- Explain the meaning of line 37 in class Complex.
- At the line of code 10 in class ComplexDemo, this statement print out a Complex object that value of attribute are displayed.

Guidelines:

- Step1. Open Eclipse IDE
- Step2. Create a new project named **Exercise0303**
- Step3: Create two classes in turn named **Complex** and **ComplexDemo** in this project

Complex class source code:

```
1.  package exercise;
2.  public class Complex {
3.    private double realPart;
4.    private double imaginaryPart;
5.    public Complex() {
6.            this.realPart = 0.0;
7.            this.imaginaryPart = 0.0;
```

```
8.          System.out.println("Complex without param!");
9.  }
10. public Complex(double realPart, double imaginaryPart) {
11.         this();
12.         this.realPart = realPart;
13.         this.imaginaryPart = imaginaryPart;
14.         System.out.println("Complex with two params!");
15. }
16. public double getRealPart() {
17.         return realPart;
18. }
19. public void setRealPart(double realPart) {
20.         this.realPart = realPart;
21. }
22. public double getImaginaryPart() {
23.         return imaginaryPart;
24. }
25. public void setImaginaryPart(double imaginaryPart) {
26.         this.imaginaryPart = imaginaryPart;
27. }
28. /**
29. *This method will find the sum of the current complex number and
30. *the passed complex number.
31. *
32. * @param otherNumber
33. *@return a new Complex number which is the sum of the two.
34. */
35. public Complex add(Complex otherNumber) {
36.         double resultRealPart = this.realPart + otherNumber.realPart;
37.         double resultImaginaryPart = this.imaginaryPart +
38.                                                 otherNumber.imaginaryPart;
39.         Complex resultNumber = new Complex(resultRealPart, resultImaginaryPart);
40.         return resultNumber;
41. }
42. @Override
43. public String toString() {
44.         return "Complex [realPart=" + realPart + ", imaginaryPart="+
45.                                                 imaginaryPart + "]";
```

```
46.   }
47.   }
48. }
```

ComplexDemo class source code:

```
1.  package exercise;
2.  public class ComplexDemo {
3.      public static void main(String[] args) {
4.          // Create two instances of Complex class
5.          Complex currentNumber = new Complex(1000, 1200);
6.          Complex otherNumber = new Complex(600, 800);
7.          // Call add method
8.          Complex resultNumber = currentNumber.add(otherNumber);
9.          // Displays result
10.         System.out.println(resultNumber);
11.     }
12.     }
13. }
```

» Step4: How to run

In Eclipse window I select **Run Test** or right-click **Run as..**:

Outputs

```
Complex without param!
Complex with two params!
Complex without param!
Complex with two params!
Complex without param!
Complex with two params!
Complex [realPart=1600.0, imaginaryPart=2000.0]
```

Bài tập 1: Xây dựng lớp SmartPhone, bao gồm

Class	SmartPhone
Properties	Tên điện thoại, hãng sản xuất, bộ nhớ RAM, giá tiền.
Constructor	0 tham số: Tên điện thoại = *"NULL"*, Hãng SX = *"Unkown"*, RAM = 0, Giá tiền = 0. 4 tham số: Nội dung biến được gán từ 4 tham số do người dùng truyền vào.
Methods	inputPhone(): Nhập thông tin điện thoại. showInfo(): Hiển thị thông tin điện thoại. comparePhone(SmartPhone): Truyền tham số điện thoại, so sánh giá tiền của 2 điện thoại và trả về nội dung tương ứng: *"Điện thoại s1 ngang giá/mắc hơn/rẻ hơn với điện thoại s2."*

Lớp SmartPhoneManagement chứa phương thức main()

- Tạo 2 đối tượng SmartPhone s1 và s2.
- Đối tượng s1 sử dụng constructor 0 tham số và gọi phương inputPhone() để truyền thông tin.
- Đối tượng s2 sử dụng constructor 4 tham số để truyền thông tin.
- So sánh đối tượng s1 và s2 và hiển thị kết luận.

Bài tập 2: Xây dựng lớp TuGiac, bao gồm

Class	Triangle
Properties	Cạnh a, cạnh b.
Constructor	0 tham số: Mặc định, cạnh a và b có giá trị bằng 0. 1 tham số: truyền vào 1 tham số cho cả 2 cạnh a và b. 2 tham số: truyền vào 2 tham số cho 2 cạnh a và b.
Methods	veTuGiac(): Xét điều kiện: - Cạnh a=0, b=0: Không thể vẽ hình. - Cạnh a≠0, b≠0: Vẽ hình bằng ký hiệu hình * với kích thước tương ứng. tinhChuVi: trả về kết quả: (a + b)*2. tinhDienTich: trả về kết quả a*b. phanLoaiTG(): phương thức trả về tên hình tứ giác - Vuông: nếu a≠0, b≠0, a=b. - Chữ nhật: nếu a≠0, b≠0, a≠b. - Không biết: nếu a=0, b=0.

Lớp TuGiacManagement chứa phương thức main()

- Tạo 3 đối tượng TuGiac t1, t2, t3.
- Đối tượng t1 sử dụng constructor 0 tham số.
- Đối tượng t2 sử dụng constructor 1 tham số.
- Đối tượng t3 sử dụng constructor 2 tham số.
- 3 đối tượng t1, t2, t3 lần lượt gọi phương thức veTuGiac, tinhChuVi, tinhDienTich, phanLoaiTG.
- 3 đối tượng xuất lần lượt xuất kết luận: *"Hình ... có chu vi là: ..., diện tích là: ..."*

Bài tập 3: Xây dựng lớp PhuongTrinh, bao gồm

Class	PhuongTrinh
Propeties	Hệ số a, hệ số b, hệ số c, delta, nghiệm x1, nghiệm x2.
Constructor	2 tham số: truyền vào tham số hệ số a, b. *(ax + b = 0)* 3 tham số: truyền vào tham số hệ số a, b, c. *($ax^2 + bx + c = 0$)*
Methods	timDelta(): Tính delta bằng công thức: $b^2 – 4 * a * c$. giaiPTBacI(): Xét điều kiện: • Hệ số a=0, b=0: Hàm trả về -1. • Hệ số a=0, b≠0: Hàm trả về 0. • Hệ số a≠0: Hàm trả về 1, x1 = x2 = -b / a. giaiPTBacII(): Xét điều kiện: • delta < 0: Hàm trả về 0. • delta = 0: Hàm trả về 1, x1 = x2 = -b / 2*a. • delta > 0: Hàm trả về 2 $x1 = \frac{-b-\sqrt{delta}}{2a}, x2 = \frac{-b+\sqrt{delta}}{2a}$. ketLuan(): Xét điều kiện từ kết quả trả về của hàm giaiPT: • Kết quả = -1: Phương trình vô số nghiệm. • Kết quả = 0: Phương trình vô nghiệm. • Kết quả = 1: Phương trình có nghiệm x = … • Kết quả = 2: Phương trình có 2 nghiệm phân biệt x1 = …, x2 = … .

Lớp PhuongTrinhManagement chứa phương thức main()

- Tạo 2 đối tượng PhuongTrinh pt1, pt2.
- Đối tượng pt1 sử dụng constructor 2 tham số và gọi hàm giaiPTBacI, KetLuan để xem kết quả
- Đối tượng pt2 sử dụng constructor 3 tham số và gọi hàm giaiPTBacII, KetLuan để xem kết quả

Bài tập 4: Xây dựng lớp PhanSo, bao gồm

Class	PhanSo
Properties	Tử số, mẫu số.
Constructor	0 tham số: Mặc định, Tử số và mẫu số có giá trị là 1. 2 tham số: Truyền vào 2 tham số lần lượt cho tử số và mẫu số.
Methods	congPhanSo(): Truyền vào một phân số, thực hiện cộng Phân số hiện hành và Phân số truyền vào rồi trả kết quả về kiểu PhanSo. rutGonPhanSo(): Phương thức thực hiện rút gọn phân số hiện hành và trả về kết quả kiểu PhanSo. hienThiPhanSo(): Phương thức trả về nội dung phân số theo cú pháp: Tử số / mẫu số.

Lớp PhanSoManagement chứa phương thức main()

- Tạo 2 đối tượng PhanSo p1, p2.
- Đối tượng p1, p2 sử dụng constructor 2 tham số truyền vào dữ liệu tử số và mẫu số.
- Tạo đối tượng PhanSo p3.
- Đối tượng p1 gọi phương thức congPhanSo truyền vào tham số p2.
- Sử dụng đối tượng p3 để hứng kết quả từ phương thức congPhanSo trên.
- Đối tượng p3 gọi phương thức rutGonPhanSo() và sau đó gọi phương thức hienThiPhanSo()