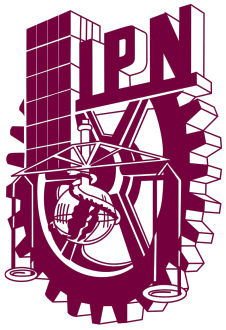


Instituto Politécnico Nacional

Escuela Superior de Computo

Práctica 5
Sobrecargade constructores, métodosy operadores

Erik Morales López

Grupo: 3CM3

Paradigmas de Programación

Docente: Miriam Pescador Rojas

Índice

1. Objetivo

Aplicar los conceptos de sobrecarga de métodos, constructores y operadores en diferentes casos prácticos.

2. Marco Teórico

2.1. Sobrecarga de métodos

En programación orientada a objetoses posible definir dos o más métodos,dentro de la misma clase,que compartan el mismo nombrepero las declaraciones de sus parámetrosdeben ser diferentes. A esto es a lo que se conoce como Sobrecarga de Métodos.

2.1.1. Reglas para sobrecargar métodos

- Los métodos sobrecargados deben de cambiar la lista de argumentos.
- Pueden cambiar el tipo de retorno.Pueden cambiar el modificador de acceso.
- Pueden declarar nuevas o más amplias excepciones.
- Un método puede ser sobrecargado en la misma clase o en una subclase.

El concepto de sobrecarga de métodos también se utilizará en Polimorfismo.

2.2. Constructores

Nos ayuda a instanciar un objeto de la clase respectiva, puede asignar valores por defecto a los atributos o por parámetro.

3. Figuras geométricas

El programa consiste en el diseño de una clase llamada figura que, a partir de una cantidad de puntos dada, debe de mostrar el tipo de figura entre cuadrado, rectángulo, circulo o triangulo y su respectiva área y perímetro. Además, debe de verificar que los puntos no sean colineales.

3.1. Clase Punto

Punto
-coordX: double -coordY: double
+ Punto() + Punto(x:double, y:double) + getX(): double + setX(x:double): void + getY(): double + setY(y:double): void

3.2. Clase Figura

Figura
-nombre: String -area: double -perimetro: double
+Figura(p1:Punto, radio:double) +Figura(p1:Punto, p2:Punto) +Figura(p1:Punto, p2:Punto,p3:Punto) -distPuntos(p1:Punto, p2:Punto) -colineales(p1:Punto, p2:Punto,p3:Punto) -colineales(p1:Punto, p2:Punto) +imprimirInformacion():void

3.3. Funcionamiento del programa

A continuación se describen la implementación y funcionamiento esperado de cada método con excepción de los set's y get's de la clase Figura, ya que en Punto no se añade alguna operación especial.

3.3.1. Constructor con un punto y radio

En este caso, se recibe un punto y un double. Lo que nos indica que la figura será un círculo, por lo tanto, lo especificamos en el atributo nombre y calculamos su área y perímetro de la forma tradicional.

```
public Figura(Punto p1, double radio){
    nombre = "Circulo";
    area = Math.PI * radio * radio;
    perimetro = Math.PI * 2.0 * radio;
}
```

3.3.2. Constructor con dos puntos

En este caso recibimos dos puntos, lo que nos indica que podremos tener un cuadrado o un rectángulo; sin embargo, debemos verificar si los puntos que nos dan corresponden a las esquinas opuestas del cuadrilátero (Implementado con la función colineales), y después calculamos su alto y ancho. En caso de que sean iguales se trata de un cuadrado y en caso contrario serán un rectángulo. Tanto el área como el perímetro se calculan de la forma habitual.

```
public Figura(Punto p1, Punto p2){
    double altura = Math.abs(p2.getCoordY()-p1.getCoordY());
    double ancho = Math.abs(p2.getCoordX()-p1.getCoordX());

    if(!colineales(p1,p2)){
        if(altura == ancho){
            nombre = "Cuadrado";
            area = ancho*ancho;
            perimetro = altura * 4;
        }
        else{
            nombre = "Rectangulo";
            area = altura*ancho;
            perimetro = 2*ancho + 2*altura;
        }
    }
    else{
        nombre = "Cuadrilatero sin puntos en esquina opuesta";
        area = 0;
        perimetro = 0;
    }
}
```

3.3.3. Constructor con tres puntos

En este caso se reciben tres puntos diferentes, lo que nos indica que es un triángulo. En este caso, para verificar que los puntos no sean colineales deberemos calcular su área, en el caso que sea 0, los puntos son colineales, en caso contrario, no es así. Para esto se usa el método colineal con tres parámetros.

```
public Figura(Punto p1, Punto p2, Punto p3){
    double a,b,c,area;

    area = colineales(p1, p2, p3);
    if(area > 0){
        nombre = "Triangulo";
        a = distPuntos(p1, p2);
        b = distPuntos(p2, p3);
        c = distPuntos(p3, p1);
        perimetro = a+b+c;
        this.area = area;
    }
    else{
        nombre = "Triángulo con sus tres puntos colineales";
        this.area = 0;
        perimetro = 0;
    }
}
```

3.3.4. Distancia entre dos puntos

Dados dos puntos devuelve la distancia entre ambos usando el teorema de Pitágoras.

```
private double distPuntos(Punto p1, Punto p2){
    double x1 = p1.getCoordX();
    double y1 = p1.getCoordY();
    double x2 = p2.getCoordX();
    double y2 = p2.getCoordY();

    return Math.sqrt(Math.pow(x1-x2, b: 2) + Math.pow(y1-y2, b: 2));
}
```

3.3.5. Colineales con dos puntos

Este método nos dice si los dos puntos dados se encuentran en esquinas opuestas con el fin de calcular del área y perímetro de los cuadriláteros. Para esto, se toman las coordenadas x,y y se verifica que x1 y x2 sean diferentes, así como y1 y y2.

```
private boolean colineales(Punto p1, Punto p2){
    boolean res;
    double x1,y1,x2,y2;

    x1 = p1.getCoordX();
    y1 = p1.getCoordY();
    x2 = p2.getCoordX();
    y2 = p2.getCoordY();

    if((x1==x2) || (y1==y2))
        res = true;
    else
        res = false;

    return res;
}
```

3.3.6. Colineales con tres puntos

Para saber si tres puntos son colineales, se calcula el área de un triangulo mediante un determinante, en caso de que el área sea diferente de 0, los puntos no son colineales.

```
private double colineales(Punto p1, Punto p2, Punto p3){
    double x1,y1,x2,y2,x3,y3,colTest;

    x1 = p1.getCoordX();
    y1 = p1.getCoordY();
    x2 = p2.getCoordX();
    y2 = p2.getCoordY();
    x3 = p3.getCoordX();
    y3 = p3.getCoordY();

    colTest =   x1*(y2-y3) +
                x2*(y3-y1) +
                x3*(y1-y2);


    return Math.abs(colTest*0.5);
}
```

3.3.7. Función main

Para probar todos las funciones implementadas se crea un arreglo con distintas figuras, asegurando que se prueban todos los constructores definidos.

```
public class Principal {
    Run | Debug
    public static void main(String args[]){
        Figura[] figs = new Figura[6];
        Punto p1,p2,p3,p4,p5,p6,p7;

        p1 = new Punto();
        p2 = new Punto(x: 5.6,y: 1.6);
        p3 = new Punto(x: 8.9,y: 7.2);
        p4 = new Punto(x: 6.0,y: 4.1);
        p5 = new Punto(x: 12.6,y: 14.8);
        p6 = new Punto();
        p7 = new Punto();

        figs[0] = new Figura();
        figs[1] = new Figura(p1,radio: 18.5);
        figs[2] = new Figura(p2,p3);
        figs[3] = new Figura(p4,p5,p1);
        figs[4] = new Figura(p1,p6);
        figs[5] = new Figura(p1,p6,p7);

        for(int i=0; i<figs.length;i++){
            figs[i].imprimirInformacion();
            System.out.println();
        }
    }
}
```

3.4. Pruebas del funcionamiento

Como se puede observar en el main, hemos creado 4 figuras con puntos no colineales, una de cada tipo (rectángulo, cuadrado, circulo y triangulo), de las cuales se calculan su área y perímetro. Por otro lado, definimos dos figuras con puntos colineales.

Así, en ambos casos se nos devuelve la respuesta esperada.

```
Nombre: Cuadrado
Área: 100.0
Perimetro: 40.0

Nombre: Circulo
Área: 1075.2100856911068
Perimetro: 116.23892818282235

Nombre: Rectangulo
Área: 18.480000000000004
Perimetro: 17.8

Nombre: Triangulo
Área: 1.9299999999999997
Perimetro: 25.158844806323888

Nombre: Cuadrilatero sin puntos en esquina opuesta
Área: 0.0
Perimetro: 0.0

Nombre: Triángulo con sus tres puntos colineales
Área: 0.0
Perimetro: 0.0
```

4. Simulación de cajero

Este programa implementa una clase llamada Cuenta, la cual es capaz de realizar operaciones bancarias básicas como retiros, consultas de saldo, transferencias, etc.

4.1. Clase Cuenta

Cuenta
-numero : long -nip : String-saldo : double -interesAnual: double -titular: String
+Cuenta() +Cuenta(num:int, ni:String, sal:double, interes:double, tit:String) +getNumero():int +setNumero(num:int):void +getNip(:String +setNip(ni:String):void +getSaldo():double +setSaldo(sal: double):void +getInteres():double +setInteres(interes:double):void +getTitular():String +setTitular(tit:String):void +deposito(cantidad:double):void +operacion(cantidad:double):boolean +operacion():double +operacion(nuevoNip:String):boolean -caracteresDiferentes(cad:String):boolean +operacion(destino:Cuenta, cantidad:double):boolean +imprimirInformacion():void

4.2. Funcionamiento básico del programa

A continuación se describen la implementación y funcionamiento esperado de cada método con excepción de los set's y get's de la clase Figura, ya que en Punto no se añade alguna operación especial.

4.2.1. Deposito

Recibe la cantidad de dinero a ingresar y se aumenta la cantidad de dinero que tiene la cuenta.

```
public void deposito(double cantidad){
    saldo+=cantidad;
}
```

4.2.2. +operacion(cantidad:double):boolean

Esta función simula el retiro de cierta cantidad de dinero, para esto, debemos verificar que la cuenta tenga el saldo suficiente para la operación. En caso positivo, se resta la cantidad retirada.

```java
public boolean operacion(double cantidad){
    boolean res = false;

    if(cantidad > 0 && cantidad <= saldo){
        saldo-=cantidad;
        res = true;
    }

    return res;
}
```

4.2.3. +operacion():double

Esta operación imprime y devuelve el saldo disponible en la cuenta bancaria.

```java
public double operacion(){
    System.out.println("Su saldo es: " + saldo);
    return saldo;
}
```

4.2.4. +operacion(nuevoNip:String):boolean

Este método simula el cambio del nip, para esto debemos verificar que sea un numero de cuatro cifras y que no tenga números repetidos. Para esto se usa la función matches, que revisa que la cadena ingresada solamente tenga los caracteres elegidos, en este caso indicamos que sean del 0 al 9. Además, se implementó la función caracteresDiferentes la cual valida la segunda condición que debe tener el nip (Se describe más adelante).

Entonces, en caso de que se validen las condiciones anteriores, se actualizara el nip y se devolverá un true como validación.

4.2.5. -caracteresDiferentes(cad:String):boolean

Esta función se encarga de hacer una transferencia a otra cuenta. Para esto, se debe validar que tenemos un saldo suficiente. En caso positivo, se modifica el saldo de ambas cuentas, en una se resta la cantidad transferida y en la otra se suma.

```java
private boolean caracteresDiferentes(String cad){
    boolean res = false;
    Set<Character> repeticiones = new HashSet<Character>();

    for(char c: cad.toCharArray()){
        repeticiones.add(c);
    }

    if(repeticiones.size() == cad.length()){
        res = true;
    }

    return res;
}
```

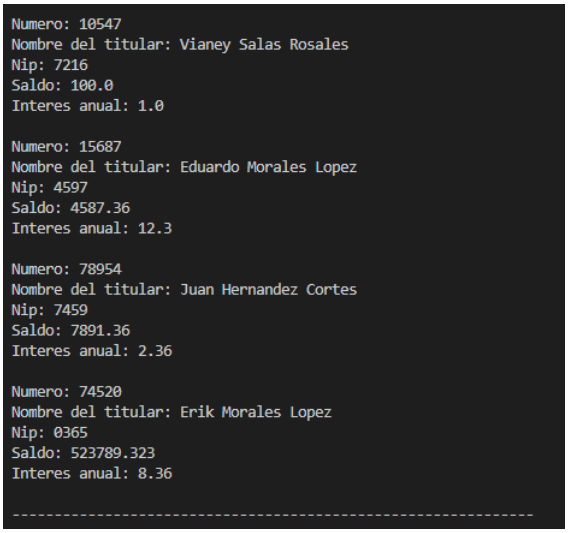
4.2.6. +operacion(destino:Cuenta, cantidad:double):boolean

Para la validar que el nuevo nip tenga 4 números diferentes se utilizara un set o conjunto, esta estructura de datos guarda elementos sin repetirlos, entonces si guardamos todos los caracteres y obtenemos la cantidad de elementos en el conjunto y lo comparamos con el tamaño original de la cadena, si son iguales, significa que cada elemento es diferente.

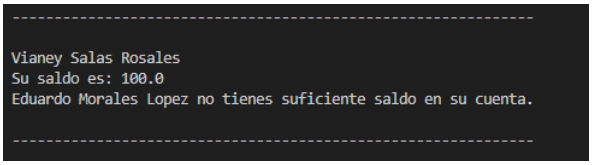
```
public boolean operacion(Cuenta destino, double cantidad){
    boolean res = false;

    if(cantidad <= saldo){
        saldo-=cantidad;
        destino.deposito(cantidad);
        res = true;
    }

    return res;
}
```

4.3. Muestra del funcionamiento

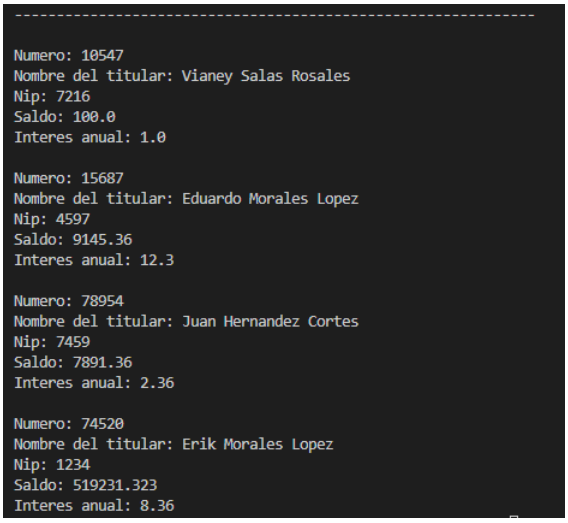
En primer lugar creamos diferentes objetos de tipo cuenta usando los constructores y métodos programados.



En este caso, imprimimos el sado de una cuenta e intentamos hacer un retiro de una cantidad mayor a la disponible en la cuenta.



Por último, realizamos el cambio de nip en diferentes cuentas y hacemos transferencias de dinero entre cuentas.



5. Conclusiones

Cuando modelamos objetos de la realidad, casi siempre queremos modelar diferentes problemas que a su vez tienen diferentes variables. Por ende, el uso de sobrecarga de métodos nos facilita esta tarea. Por un lado, nos permite tener una mejor organización del código y por otro, podemos adaptar la realidad de una forma mas sencilla.