Universidad Nacional de Colombia. Sede Bogotá.

Programación Orientada a Objetos

Taller III

Integrantes: Castro Sergio, Isea Raúl, Ortíz Jessica

1) Responda las siguientes preguntas:

a) ¿Cuál cree que es rol de herencia en un programa de Java?

La herencia es una forma de reutilización de software en la que se crea una nueva clase al absorber los miembros de una existente, y se mejoran con nuevas capacidades, o con modificaciones en las capacidades ya existentes. Con la herencia, los programadores ahorran tiempo durante el desarrollo, al basar las nuevas clases en el software existente, probado y depurado, de alta calidad. Esto también aumenta la probabilidad de que un sistema se implemente y mantenga con efectividad.

Al crear una clase, en vez de declarar miembros completamente nuevos, el programador puede designar que la nueva clase herede los miembros de una existente. La cual se conoce como superclase, y la clase nueva como subclase. Una subclase puede agregar sus propios campos y métodos. Java, sólo soporta la herencia simple, en donde cada clase se deriva sólo de *una* superclase directa.

b) ¿Cómo la herencia promueve la reutilización de software?

La programación orientada a objetos facilita la reutilización de software, con lo que comúnmente se obtiene una potencial reducción en el tiempo de desarrollo. La disponibilidad de bibliotecas de clases extensas y útiles produce los máximos beneficios de la reutilización de software a través de la herencia. Las bibliotecas de clases estándar que se incluyen con Java tienden a ser de propósito general, y fomentan mucho la reutilización de software. Existen muchas otras bibliotecas de clases.

c) ¿Cómo se podría explicar la Jerarquía (Hierarchy) en la programación orientada a objetos?

La jerarquía de datos se puede explicar como un conjunto de elementos de datos que se procesan por un computador. Esta jerarquía está representada por una compleja estructura que se hace cada vez más grande. La jerarquía de datos se encuentra sintetizada por niveles, los cuales se explican a continuación:

i. Bits: El elemento de datos más pequeño en una computadora puede asumir el valor 0 o el valor 1. A dicho elemento de datos se le denomina bit (abreviación de “dígito binario”: un dígito que puede asumir uno de dos valores).

ii. Caracteres: el conjunto de caracteres de la computadora es el conjunto de todos los que se utilizan para escribir programas y representar elementos de datos. Las computadoras sólo procesan los 1 y los 0, por lo que el conjunto de caracteres de una computadora representa a cada uno como un patrón de los 1 y los 0. Java usa caracteres Unicode® que están compuestos de dos bytes, cada uno de los cuales está formado a su vez de ocho bits.

iii. Campos: Así como los caracteres están compuestos de bits, los campos lo están por caracteres o bytes. Un campo es un grupo de caracteres o bytes que transmiten un significado. Por ejemplo, un campo compuesto de letras mayúsculas y minúsculas se puede usar para representar el nombre de una persona, y uno compuesto de dígitos decimales podría representar su edad.

iv. Registros: Se pueden usar varios campos relacionados para componer un registro (el cual se implementa como una el ase en Java). Por ejemplo, en un sistema de nómina, el registro de un empleado podría consistir en los siguientes campos (los posibles tipos para éstos se muestran entre paréntesis): número de identificación del empleado (un número entero), nombre (una cadena de caracteres), dirección (una cadena de caracteres), salario por horas (un número con punto decimal), ingresos del año a la fecha (un número con punto decimal), monto de impuestos retenidos (un número con punto decimal).
Así, un registro es un grupo de campos relacionados. En el ejemplo anterior, todos los campos pertenecen al mismo empleado. Una compañía podría tener muchos empleados y un registro de nómina para cada uno.

v. Archivo: Un archivo es un grupo de registros relacionados. Dicho en forma más general, un archivo contiene datos arbitrarios en formatos arbitrarios. En algunos sistemas operativos, un archivo se ve tan sólo como una *secuencia de bytes*: cualquier organización de esos bytes en un archivo, como cuando se organizan los datos en registros, es una vista creada por el programador de la aplicación.

d) Explique la diferencia entre Composición (composition) y herencia. Dé un ejemplo.

La composición y la herencia colocan sub-objetos dentro de la clase. Ambos usan la lista de inicialización del constructor para construir esos sub-objetos.

La composición generalmente se usa cuando se quieren las características de una clase existente dentro de su clase, pero no en su interfaz. Esto es, aloja un objeto para implementar características en su clase, pero el usuario de su clase ve el interfaz que se ha definido, en vez del interfaz de la clase original. Para hacer esto, se sigue el típico patrón de alojar objetos privados de clases existentes en su nueva clase.

Pero ¿qué ocurre si se quiere que todo funcione cómo la clase deseada? A eso se le llama subtipos porque está creando un nuevo tipo desde uno ya existente y lo que se quiere es que el nuevo tipo tenga la misma interfaz que el tipo existente (además de otras funciones que se deseen añadir) para que se pueda utilizar en cualquier lugar donde se utilizaba el tipo existente. Aquí es dónde la herencia es esencial.

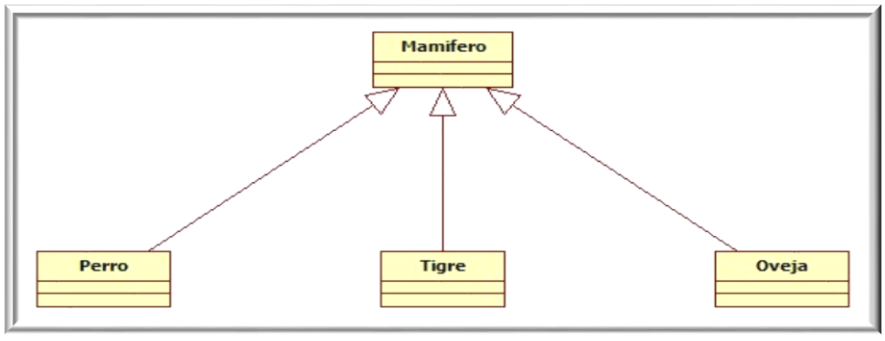


Figura 1. Ejemplo de herencia.

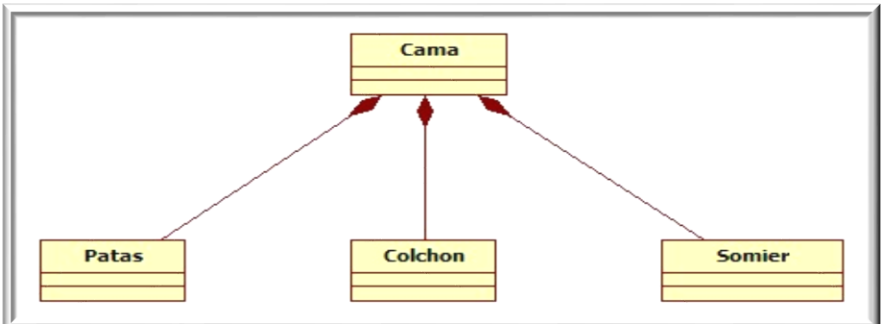


Figura 2. Ejemplo de composición.

e) En java una subclase puede heredar de máximo una superclase. En otros lenguajes como C++ es posible que una clase herede de más de una clase (Herencia múltiple). Explique los pros y contras de esta práctica.

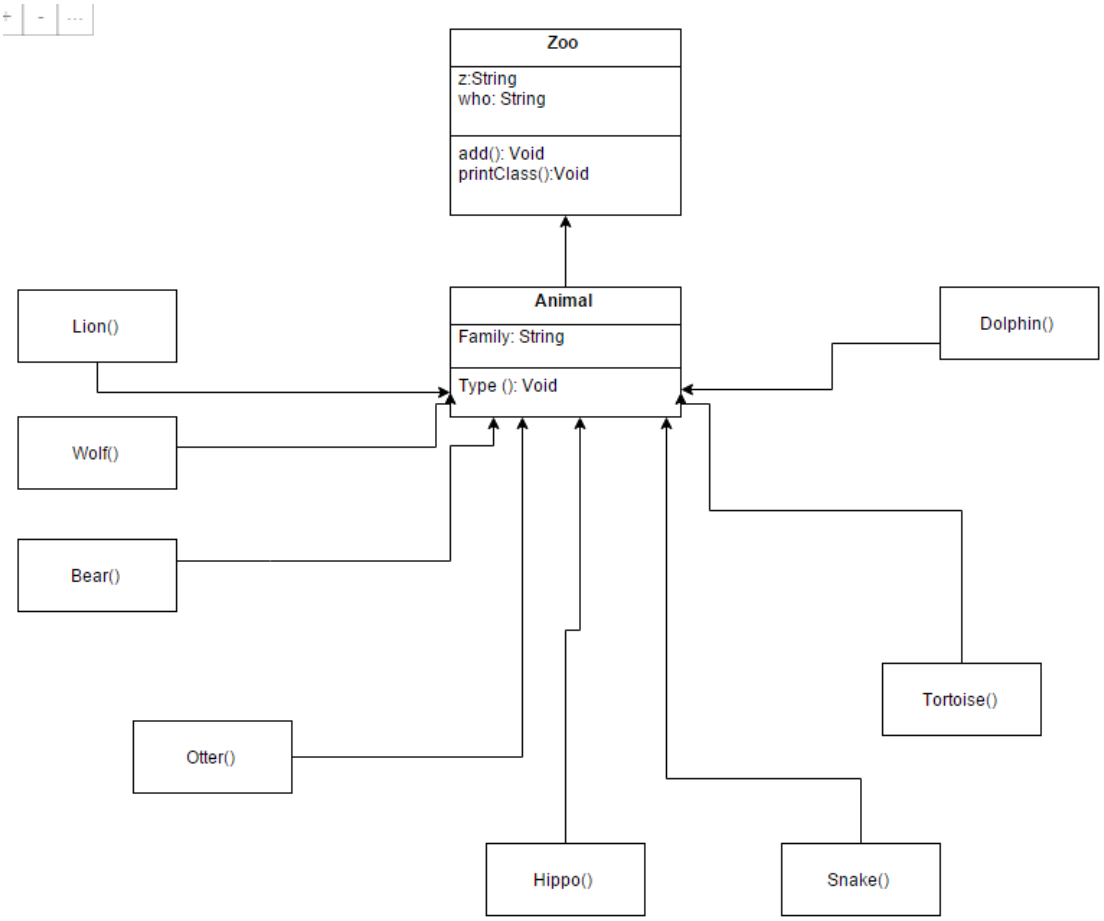
Ventajas: la herencia múltiple es más apropiada para definir objetos compuestos o una combinación de objetos. Mientras que la herencia simple es más útil para definir objetos que son más especializaciones de objetos generales.

Desventajas: el problema de la herencia múltiple está en que puede introducir más ambigüedades. La ambigüedad se produce cuando se heredan dos clases base, que a su vez contienen ambas a la misma clase indirecta, produciéndose la circunstancia de que la nueva clase base contiene dos veces los objetos de la clase base indirecta.

2) Herencia

a. Zoo-Diseño Top-Down

Usted ha sido contactado por el zoológico para diseñar un simulador que sirva para estudiar el reino animal. Usted debe crear un diagrama de clases que represente a los animales del zoológico. Cada animal va a ser representado como un objeto. Use todos los atributos y métodos que considere relevantes para el modelado.




```
import java.util.*;
class Animal {

   String Family;          //instance data
   Animal(){               //constructor, sets data
      Family = "";
   }
   public String Type(){     //  method, uses data
      return Family;
   }
}

class Lion extends Animal {    // class inheritence
```

```
    Lion(){
        Family = "Cats";
    }
}

class Wolf extends Animal {

    Wolf(){
        Family = "Canines";
    }
}

class Bear extends Animal {

    Bear(){
        Family = "Ursidae";
    }
}

class Otter extends Animal {

    Otter(){
        Family = "Musteliade";
    }
}

class Hippo extends Animal {

    Hippo(){
        Family = "Hippopotamidae";
    }
```

```
}

class Snake extends Animal {

    Snake(){
        Family = "Serpentes";
    }
}

class Tortoise extends Animal {

    Tortoise(){
        Family = "Testudines";
    }
}

class Dolphin extends Animal {

    Dolphin(){
        Family = "Delphinidae";
    }
}

class Zoo {

    public static ArrayList<Animal> animals=new ArrayList<>();

    void add(Animal who) {
            animals.add(who);
    }

    static void println(String message){
```

```
        System.out.println(message);
    }

    void printClass(){
        println("The animals in the zoo are:  ");
        for (Animal who : animals){
            println(who.Type());
        }
    }

    public static void main(String[] args) {
        Zoo z = new Zoo();
        z.add(new Lion());
        z.add(new Wolf());
        z.add(new Bear());
        z.add(new Otter());
        z.add(new Hippo());
        z.add(new Snake());
        z.add(new Tortoise());
        z.add(new Dolphin());
        z.printClass();
        /**println("");
        println("The first animal Type '" + Zoo.animals.get(0).Type() + "'");
        println("The last Type '" + Zoo.animals.lastElement().Type() + "'");*/
    }
}
```

b. Figuras-Generalización

Usted está trabajando desarrollando software de enseñanza para niños, Ahora se encuentra en un proyecto el cual busca enseñar acerca de diferentes figuras geométricas. El director del proyecto le entrega los requerimientos del producto, en este documento de requerimientos se especifica que se trabaja con un número limitado de figuras. Rectángulo, paralelogramo, triangulo rectángulo, triangulo isósceles.

```
public class GeometricFigures {
}

class Rectangle extends GeometricFigures {
private double width=1.0;
private double length=1.0;

public Rectangle() {
}

public Rectangle(double width, double length) {
    this.width = width;
    this.length = length;
}
//Return width
public double getWidth() {
    return width;
}

//Set a new width
public void setWidth(double width) {
    this.width = width;
}

// Return length
public double getLenght() {
    return length;
}

// Set a new length
public void setLenght(double length) {
    this.length = length;
```

```
}

// Return area
public double getArea() {
    return width * length;
}

// Return perimeter
public double getPerimeter() {
    return 2 * (width * length);
}
}
class RectangleTriangle extends GeometricFigures {
private double base=1.0;
private double height=1.0;

public RectangleTriangle() {
}

public RectangleTriangle(double base, double height) {
    this.base = base;
    this.height = height;
}
// Return width
public double getBase() {
    return base;
}
// Set a new width
public void setBase(double base) {
    this.base = base;
}
//Return height
```

```
public double getHeight() {
    return height;
}

// Set a new height
public void setHeight(double height) {
    this.height = height;
}

// Return area
public double getArea() {
    return ((base * height)/2);
}

// Return perimeter
public double getPerimeter() {
    return (base+height+Math.sqrt(base*base+height*height));
}
}
class IsocelesTriangle extends GeometricFigures {
private double base=1.0;
private double height=1.0;

public IsocelesTriangle() {
}

public IsocelesTriangle(double base, double height) {
    this.base = base;
    this.height = height;
}
//Return width
public double getBase() {
```

```
    return base;
}
// Set a new width
public void setBase(double base) {
    this.base = base;
}

//Return height
public double getHeight() {
    return height;
}

//Set a new height
public void setHeight(double height) {
    this.height = height;
}

// Return area
public double getArea() {
    return ((base * height )/2);
}
// Return perimeter
public double getPerimeter() {
    return (base+height+Math.sqrt(base*base+height*height));
}
}

class Paralelogram extends GeometricFigures {
private double width=1.0;
private double length=1.0;

public Paralelogram() {
```

```
}

public Paralelogram(double width, double length) {
    this.width = width;
    this.length = length;
}

//Return width
public double getWidth() {
    return width;
}

// Set a new width
public void setWidth(double width) {
    this.width = width;
}
// Return height
public double getLenght() {
    return length;
}
//Set a new height
public void setLenght(double length) {
    this.length = length;
}
//Return area
public double getArea() {
    return width * length;
}
// Return perimeter
public double getPerimeter() {
    return 2 * (width * length);
}
```

```
}
class TestFigures {
public static void main(String[] args) {

   Rectangle rectangle =new Rectangle(2, 4);
   System.out.println("Rectangle ");
   System.out.println("The area is " + rectangle.getArea());
   System.out.println("The perimeter is " + rectangle.getPerimeter());

   RectangleTriangle rectangletriangle =new RectangleTriangle(2, 4);
   System.out.println("   ");
   System.out.println("Rectangletriangle ");
   System.out.println("The area is " + rectangletriangle.getArea());
   System.out.println("The perimeter is " + rectangletriangle.getPerimeter());

   IsocelesTriangle isocelestriangle =new IsocelesTriangle(2, 4);
   System.out.println("   ");
   System.out.println("Isocelestriangle ");
   System.out.println("The area is " + isocelestriangle.getArea());
   System.out.println("The perimeter is " + isocelestriangle.getPerimeter());

   Paralelogram paralelogram =new Paralelogram(2, 4);
   System.out.println("   ");
   System.out.println("Paralelogram ");
   System.out.println("The area is " + paralelogram.getArea());
   System.out.println("The perimeter is " + paralelogram.getPerimeter());
}
}
```

¿Cuáles características tienen en común?, ¿Cómo se generarían las clases más generales?

Los métodos que usa para generar los perímetros y áreas, así como los get y set de los atributos son los mismos. Se pueden unificar las clases que representan a las figuras.

Clase TestShapes

```
public class TestShapes
{
double width=1.0;
double length=1.0;
double height=1.0;
double base=1.0;
}

class Rectangles extends TestShapes{
public Rectangles() {
}
public Rectangles(double width, double length) {
    this.width = width;
    this.length = length;
}

// Return width
public double getWidth() {
    return width;
}

// Set a new width
public void setWidth(double width) {
    this.width = width;
}

// Return length
public double getLenght() {
    return length;
}

// Set a new length
```

```
public void setLenght(double length) {
    this.length = length;
}

// Return area
public double getArea() {
    return width * length;
}

// Return perimeter
public double getPerimeter() {
    return 2 * (width * length);
}
}

class Triangles extends TestShapes {

public Triangles() {
}

public Triangles(double base, double height) {
    this.base = base;
    this.height = height;
}

// Return width
public double getBase() {
    return base;
}

//Set a new width
public void setBase(double base) {
```

```
    this.base = base;
}

//Return height
public double getHeight() {
    return height;
}

//Set a new height
public void setHeight(double height) {
    this.height = height;
}

// Return area
public double getArea() {
    return ((base * height)/2);
}

// Return perimeter
public double getPerimeter() {
    return (base+height+Math.sqrt(base*base+height*height));
}
}

class Test{
public static void main(String[] args) {

    Rectangles rectangles =new Rectangles(2, 4);
    System.out.println("Rectangle ");
    System.out.println("The area is " + rectangles.getArea());
    System.out.println("The perimeter is " + rectangles.getPerimeter());
```

```
        Triangles triangles =new Triangles(2, 4);
        System.out.println("   ");
        System.out.println("Triangles ");
        System.out.println("The area is " + triangles.getArea());
        System.out.println("The perimeter is " + triangles.getPerimeter());

    }
    }
```

c. ¿Qué se puede decir acerca de los enfoques (Generalización, Especificación)?

La generalización es necesaria solo cuando las características de las clases presentes a desarrollar en la orden de requerimientos hacen que esta sea indispensable para el desarrollo de lo que el cliente necesita, por otro lado, la especificación es más necesaria cuando las características de las clases, no ameritan que se genere herencia, es decir que todos comparten los mismos rasgos específicos que al estar por separado hacen que el programa sea pesado y no muy eficiente en su desarrollo.

3) Mas Figuras

a) Escriba una Jerarquía de Herencia para las clases “Quadrilateral”, “Trapezoid”, “Parallelogram”, “Rectangle” y “Square”. Use “Quadrilateral” como la superclase de la herencia. Especifique los atributos y métodos de cada clase. Los atributos privados de “Quadrilateral” deben ser los pares de coordenadas para los 4 esquinas del cuadrilátero. Escriba un programa que instancie objetos de sus clases he imprima para cada objeto el área (Excepto para el Cuadrilátero).

- **Clase Point**

```
public class Point {

    private double x;
    private double y;

    public Point(){
        x = 0.0;
        y = 0.0;
    }

    public Point(double x, double y) {
        this.x = x;
```

```
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    @Override
    public String toString() {
        return "( " + x + " , " + y + " )";
    }


}
```

- **Clase Quadrilateral**

```
public class Quadrilateral {
    private Point p1;
    private Point p2;
    private Point p3;
    private Point p4;
```

```
    public Quadrilateral(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4){
        p1 = new Point(x1,y1);
        p2 = new Point(x2,y2);
        p3 = new Point(x3,y3);
        p4 = new Point(x4,y4);
    }

    public Point getP1() {
        return p1;
    }

    public void setP1(Point p1) {
        this.p1 = p1;
    }

    public Point getP2() {
        return p2;
    }

    public void setP2(Point p2) {
        this.p2 = p2;
    }

    public Point getP3() {
        return p3;
    }

    public void setP3(Point p3) {
        this.p3 = p3;
    }
```

```
        public Point getP4() {
                return p4;
        }

        public void setP4(Point p4) {
                this.p4 = p4;
        }

        public String toString(){
      return ("\nCoordinates of Quadrilateral are:\n"+ getP1() +"," + getP2() +"," + getP3() + "," +
getP4()) + "\n";
   }

}
```

- **Clase Trapezoid**

```
public class Trapezoid extends Quadrilateral{

        private double sideA;
        private double sideB;
        private double h;

        public Trapezoid(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4) {
                super(x1, y1, x2, y2, x3, y3, x4, y4);
        }

        public double getSideA() {
                return (Math.abs(getP1().getX()-getP2().getX()));          //Teniendo en
cuenta que los puntos p1 y p4 son vecinos y se distancian por el eje y
        }
```

```
        public double getSideB() {
                return (Math.abs(getP3().getX()-getP4().getX()));
                }

        public double getH() {
                return (Math.abs(getP1().getY()-getP4().getY()));
        }

        public double getArea(){
                sideA = getSideA();
                sideB = getSideB();
                h = getH();

                return (h * (sideA + sideB)) / 2;
        }

        public String toString(){
      return ("\nCoordinates of Trapezoid are:\n"+ getP1() +"," + getP2() +"," + getP3() + "," +
getP4() + "\nHeight is: " + getH() + "\nArea is: " + getArea()) + "\n";
   }

}
```

- **Clase Parallelogram**

```
public class Parallelogram extends Trapezoid{

        public Parallelogram(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4) {
                super(x1, y1, x2, y2, x3, y3, x4, y4);
        }

        public double getW(){
     return  getSideA();
   }
```

```
    public double getArea(){
        return (getW() * getH());
    }

    public String toString(){
        return ("\nCoordinates of Parallelogram are:\n"+ getP1() +"," + getP2() +"," + getP3() + "," +
getP4() + "\nWidth is: " + getW() + "\nHeight is: " + getH() + "\nArea is: " + getArea())+"\n";
    }

}
```

- **Clase Rectangle**

```
public class Rectangle extends Parallelogram{

        public Rectangle(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4) {
                super(x1, y1, x2, y2, x3, y3, x4, y4);
        }

        public double getArea(){
                return getW() * getH();
        }

        public String toString(){
        return ("\nCoordinates of Rectangle are:\n"+ getP1() +"," + getP2() +"," + getP3() + "," +
getP4() + "\nWidth is: " + getW() + "\nHeight is: " + getH() + "\nArea is: " + getArea())+"\n";
    }
}
```

- **Clase Square**

```
public class Square extends Rectangle{

        public Square(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4) {
```

```
            super(x1, y1, x2, y2, x3, y3, x4, y4);
        }

        public double getSide(){
            return getW();
        }

        public String toString(){
      return ("\nCoordinates of Square are:\n"+ getP1() +"," + getP2() +"," + getP3() + "," + getP4()
+ "\nSide is: " + getSide()  + "\nArea is: " + getArea())+"\n";
   }
}
```


Bibliografía:

[1] H.M. Deitel and P.J. Deitel, Java How to Program.

[2] – Consulta en línea -http://informatica.utem.cl/~mcast/ESDATOS/POO/herencia.pdf

[3] – Consulta en línea - http://doingbettersoftware.blogspot.com.co/2012/12/herencia-vs-composicion.