Relaciones Entre Objetos

John Jairo Montoya Gómez Código: 2879540

Juan David Balcázar Bedoya Código: 2879963

Taller

1. Responda las siguientes preguntas:

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

La herencia permite definir nuevas clases con base a clases existentes, la nueva clase hereda los atributos y el comportamiento de la clase existente, permitiendo en la nueva clase mejorar las capacidades que esta tiene y permitir reducir problemas cuando se hace funcional.

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

La herencia promueve la reutilización de software cuando este es comprobado, depurado y de alta calidad, permite ahorrar tiempo a la hora de desarrollar y tener un producto fiable, comprensible y adaptable.

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

La jerarquía es la que permite ordenar y clasificar abstracciones en una estructura de árbol, existen jerarquías de clases, de herencia, de especialización, de tipo, etc, entre esta estructura se va formando una especie de dependencia.

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

La herencia es una relación “es-un”, en la que un objeto de una subclase también puede tratarse como un objeto de su superclase.

Por ejemplo una clase Taxi o Bus puede heredar las propiedades de una clase Vehiculo.

La composición es una relación “tiene-un” en la que el objeto de una clase contiene referencias a objetos de otras clases.

Por ejemplo una clase Carro puede adquirir la característica Volante de otra clase o la característica subir o bajar ventana que tiene una clase llamada Ventana

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.

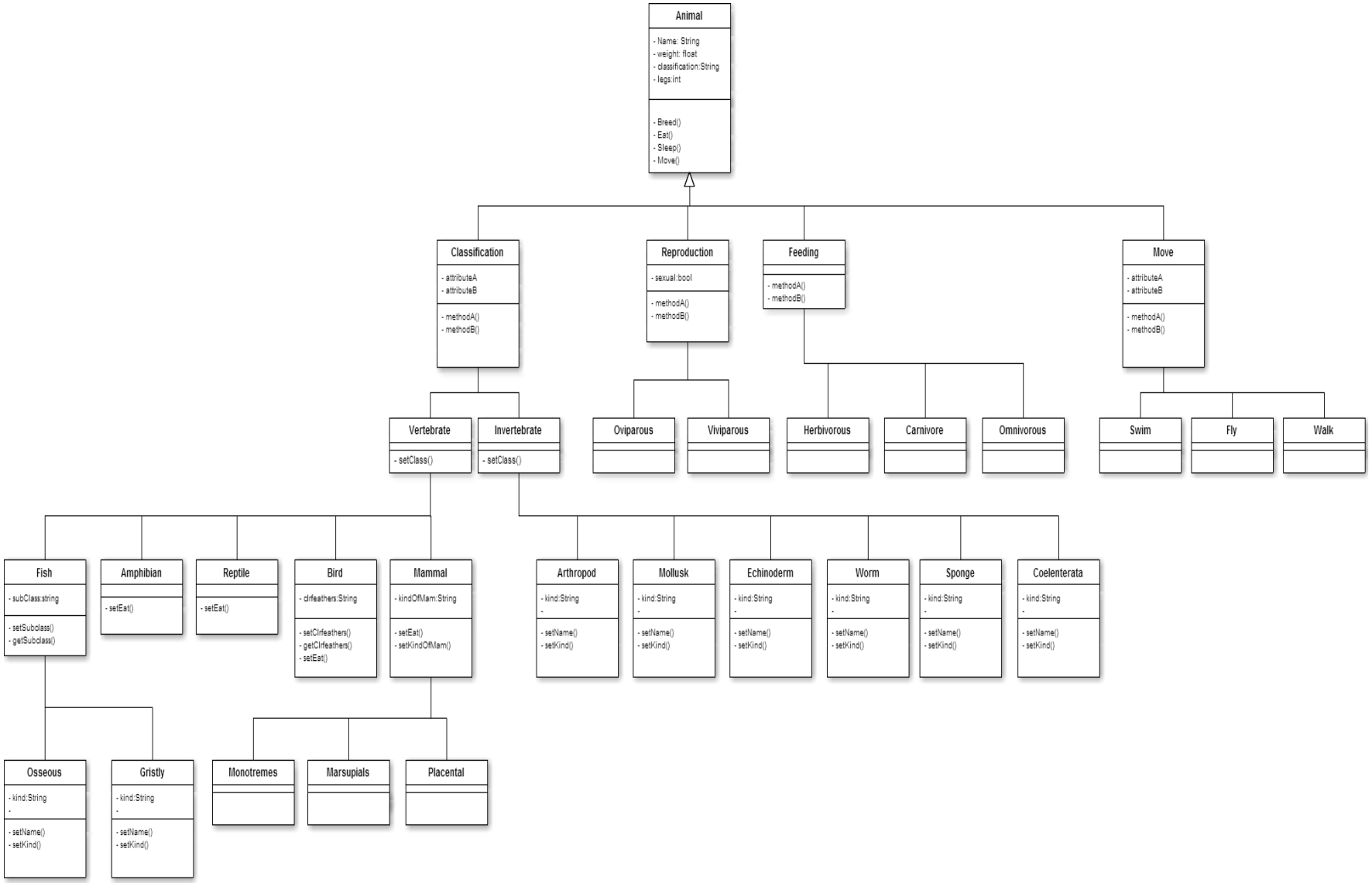
Ventajas:

- Permitir heredar funcionalidades de otras clases
- Reutilización del código, se evitan redundancias
- Permite organizar las clases.

Desventajas:

- se puede dar la duplicación de propiedades
- se puede presentar colisión de nombres
- dificultad del mantenimiento de aplicaciones
- dificultad en la extensión de las aplicaciones
- Ambigüedad y aumentada complejidad

2- a)


Animal
- Name: String
- weight: float
- classification:String
- legs:int
- Breed()
- Eat()
- Sleep()
- Move()
Classification
- attributeA
- attributeB
- methodA()
- methodB()
Reproduction
- sexual:bool
- methodA()
- methodB()
Feeding
- methodA()
- methodB()
Move
- attributeA
- attributeB
- methodA()
- methodB()
Vertebrate
- setClass()
Invertebrate
- setClass()
Oviparous
Viviparous
Herbivorous
Carnivore
Omnivorous
Swim
Fly
Walk
Fish
- subClass:string
- setSubclass()
- getSubclass()
Amphibian
- setEat()
Reptile
- setEat()
Bird
- clrfeathers:String
- setClrfeathers()
- getClrfeathers()
- setEat()
Mammal
- kindOfMam:String
- setEat()
- setKindOfMam()
Arthropod
- kind:String
-
- setName()
- setKind()
Mollusk
- kind:String
-
- setName()
- setKind()
Echinoderm
- kind:String
-
- setName()
- setKind()
Worm
- kind:String
-
- setName()
- setKind()
Sponge
- kind:String
-
- setName()
- setKind()
Coelenterata
- kind:String
-
- setName()
- setKind()
Osseous
- kind:String
-
- setName()
- setKind()
Gristly
- kind:String
-
- setName()
- setKind()
Monotremes
Marsupials
Placental

2.b Figuras – Generalizacón

```
public class Rectangle extends Shape {



	@Override
	public double getPerimeter() {
		return 2 * width + 2 * height;
	}
	@Override
	public double getArea() {
		return width * height;
	}

}

public class RectangleTriangle extends Shape {

	@Override
	public double getPerimeter(){
		double hypotenuse = Math.sqrt((base * base + height
				* height));
		return base  + height + hypotenuse;
	}

	@Override
	public double getArea() {
		return base * height / 2;
	}


}

public class Prallelogram extends Shape{


	private double parallelogramSide;

	public Prallelogram(){
		super();
		parallelogramSide = 1;
	}
	@Override
	public double getPerimeter() {
		return 2 * parallelogramSide + 2 * width;
	}

	@Override
	public double getArea() {
		return width * height;
	}
	public double getParallelogramSide(){
		return parallelogramSide;
	}
	public void setParallelogramSide(double parallelogramSide){
```

```
            this.parallelogramSide = parallelogramSide;
        }


}
public class IsocelesTriangle extends Shape {

        @Override
        public double getPerimeter() {
            double side = Math.sqrt((base / 2) * (base / 2) +
                        height * height);
            return base + 2 * side;
        }

        @Override
        public double getArea() {
            return base * height / 2;
        }

}
/*Abstract class to create all the shapes from, since all
 shapes have common attributes and methods.*/
public abstract class Shape {

        protected double width;
        protected double height;
        protected double base;
        protected double paraSide;

        public Shape(){
            width = 1;
            height = 1;
            base = 1;
        }
        public abstract double getPerimeter();
        public abstract double getArea();
        public double getWidth(){
            return width;
        }
        public double getHeight(){
            return height;
        }
        public double getBase(){
            return base;
        }
        public void setWidth(double width){
            this.width = width;
        }
        public void setHeight(double height){
            this.height = height;
        }
        public void setBase(double base){
            this.base = base;
        }
}
```

```
public class TestShapes {

	public static void main(String[] args) {

	Rectangle r = new Rectangle();
	System.out.print("Rectangle" + "\n" +
	"Area: " + r.getArea() + "\n" +
	"Perimeter: " + r.getPerimeter() + "\n" +
	"Width: " + r.getWidth() + "\n" +
	"Height: " + r.getHeight() + "\n");

	RectangleTriangle rt = new RectangleTriangle();
	System.out.print("Rectangle Triangle" + "\n" +
	"Area: " + rt.getArea() + "\n" +
	"Perimeter: " + rt.getPerimeter() + "\n" +
	"Width: " + rt.getWidth() + "\n" +
	"Height: " + rt.getHeight() + "\n");

	IsocelesTriangle it = new IsocelesTriangle();
	System.out.print("Isoceles Triangle" + "\n" +
	"Area: " + it.getArea() + "\n" +
	"Perimeter: " + it.getPerimeter() + "\n" +
	"Width: " + it.getWidth() + "\n" +
	"Height: " + it.getHeight() + "\n");

	Prallelogram p = new Prallelogram();
	System.out.print("Prallelogram" + "\n" +
	"Area: " + p.getArea() + "\n" +
	"Perimeter: " + p.getPerimeter() + "\n" +
	"Width: " + p.getWidth() + "\n" +
	"Height: " + p.getHeight() + "\n");


	}

}
```

Se crea una clase abstracta para que todas las otras figuras puedan heredar. Ya que todas las figuras tienen atributos y métodos comunes. Los métodos getArea() y getPerimeter() son abstractos ya que cada figura tiene una forma diferente de optener el área y el perímetro. El resto de métodos como getWith() and getHeight no son abstractos ya que se usan de la misma forma para todas las figuras.

c. La generalización es útil cuando se quiere pensar en un grupo de objetos que poseen características similares o comunes. La especificación nos permite formar objetos a partir de conceptos más abstractos.

3.

```
public class Point {

	public double x;
	public double y;

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

}
public class Quadrilateral {

	protected double width;
	protected double height;
	protected Point point1;
	protected Point point2;
	protected Point point3;
	protected Point point4;

	public Quadrilateral(Point p1, Point p2, Point p3, Point p4){
		point1 = p1;
		point2 = p2;
		point3 = p3;
		point4 = p4;
		width = Math.abs(point2.x - point1.x);
		height = Math.abs(point4.y - point2.y);

	}
	public double getWidth(){
		return width;
	}
	public double getHeight(){
		return height;
	}
}

public class Trapezoid extends Quadrilateral {

	private double base1;
	private double base2;

	public Trapezoid(Point p1, Point p2, Point p3, Point p4) {
		super(p1, p2, p3, p4);
		base1 = Math.abs(point3.x - point4.x);
		base2 = Math.abs(point1.x - point2.x);
	}

	public double getArea(){
		return (base1 + base2) * height / 2;
	}

}
```

```
public class Parallelogram extends Quadrilateral {

	public Parallelogram(Point p1, Point p2, Point p3, Point p4) {
		super(p1, p2, p3, p4);
	}


	public double getArea(){
		return width * height;
	}
}

public class Rectangle2 extends Quadrilateral {

	public Rectangle2(Point p1, Point p2, Point p3, Point p4) {
		super(p1, p2, p3, p4);

	}

	public double getArea(){
		return width * height;
	}

}
public class Square extends Quadrilateral{

	public Square(Point p1, Point p2, Point p3, Point p4) {
		super(p1, p2, p3, p4);

	}

	public double getArea(){
		return width * width;
	}
}




public class TestShapes {

	public static void main(String[] args) {



	//Punto 3
	Point p1 = new Point(0, 0);
	Point p2 = new Point(10, 0);
	Point p3 = new Point(8, 5);
	Point p4 = new Point(3.3, 5);
	Trapezoid t = new Trapezoid(p1, p2, p3, p4);
	System.out.println("Area of the Trapezoid: " + t.getArea() + " height: " +
t.getHeight());
	p1 = new Point(5, 5);
	p2 = new Point(11, 5);
```

```
		p3 = new Point(12, 20);
		p4 = new Point(6, 20);
		Parallelogram pl = new Parallelogram(p1, p2, p3, p4);
		System.out.println("Area of the Parallelogram: " + pl.getArea() + "
height: " + pl.getHeight()
		+ " width: " + pl.getWidth());
		p1 = new Point(17, 14);
		p2 = new Point(30, 14);
		p3 = new Point(30, 28);
		p4 = new Point(17, 28);
		Rectangle2 r2 = new Rectangle2(p1, p2, p3, p4);
		System.out.println("Area of the Rectangle2: " + r2.getArea() + " height: "
+ r2.getHeight()
		+ " width: " + r2.getWidth());
		p1 = new Point(4, 0);
		p2 = new Point(8, 0);
		p3 = new Point(8, 4);
		p4 = new Point(4, 4);
		Square s = new Square(p1, p2, p3, p4);
		System.out.println("Area of the Square: " + s.getArea() + " side: " +
s.getHeight());

		}

}
```

Todas las clases heredan de la superclase quadrilateral, que contiene atributos comunes a todas las figuras como Point, width y height. Cada clase tiene el método getArea() ya que algunas áreas se obtienen de forma distinta como la del trapezoide y el cuadrado.