Taller 3 – Relaciones entre Objetos

Programación orientada a objetos

Juan Pablo Vallejo Figueroa – 262155

1.

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

La herencia permite definir una clase tomando como base a otra clase ya existente. Esto es una de las bases de la reutilización de código, en lugar de copiar y pegar.

En java, la herencia se especifica agregando la claúsula extends después del nombre de la clase. En la claúsula extends indicaremos el nombre de la clase base de la cuál queremos heredar.

Al heredar de una clase base heredaremos tanto los atributos como los métodos, mientras que los constructores son utilizados, pero no heredados.

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

Es una herramienta de reutilización de software porque con la herencia podemos utilizar los componentes de una clase ya existente de forma total o parcial, dándonos la oportunidad, incluso, de mejorar ese código o utilizarlo tal cual. Permite la reutilización de software probado y depurado por otros programadores, disminuyendo así el tiempo del desarrollo. Para utilizar la herencia debemos crear una clase que "extienda o herede" de otra, la clase superior es conocida como *Superclase* y la clase que extiende de esta es una *Subclase*.

Una superclase, se podría decir que, es una idea general sobre un objeto y la subclase es más específica y trata con los detalles. Por consiguiente podemos afirmar que la herencia se trata de una especialización de las diferentes clases heredadas.

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

Por jerarquía denotamos el orden de relación que se produce entre abstracciones diferentes. Permite una ordenación de las abstracciones. Los tipos de jerarquía más útiles son:

- Herencia (generalización/especialización, padre/hijo, jerarquía del tipo "es un"...). Una clase (subclase) comparte la estructura o comportamiento definido en otra clase, llamada superclase.

- Herencia múltiple: Una clase comparte la estructura o comportamiento de varias superclases.

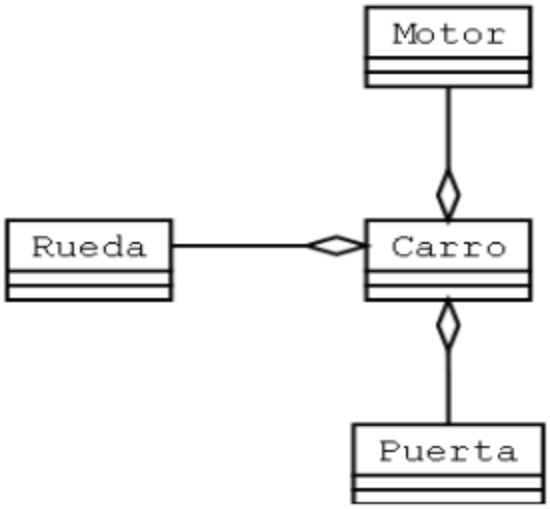
- Agregación: Comprende relaciones del tipo "es parte de" al realizar una descomposición.

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

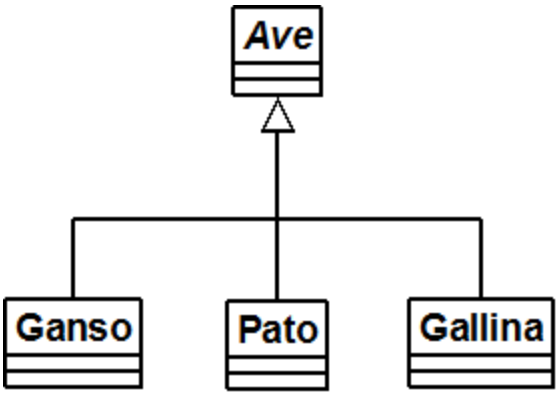
La composición sirve cuando hacen falta las características de una clase existente dentro de una nueva, pero no su interfaz. En herencia se puede hacer una versión especial de una clase existente, reutilizando su interfaz

Con la composición los objetos miembro privados pueden cambiarse en tiempo de ejecución. Los cambios en el objeto no afectan el código cliente. En la relación de herencia se define en tiempo de compilación y no puede cambiarse en tiempo de ejecución y permite reinterpretar el tipo de un objeto en un tiempo de ejecución.

Ejemplo de composición.



Ejemplo de Herencia



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.

Como java no soporta herencia multiple, a veces es favorable la composición. Se favorece debido a que se utiliza en ejemplos de patrón de diseño, donde se utiliza la composición y delegación para cambiar el comportamiento del contexto, sin tocar el código contexto.

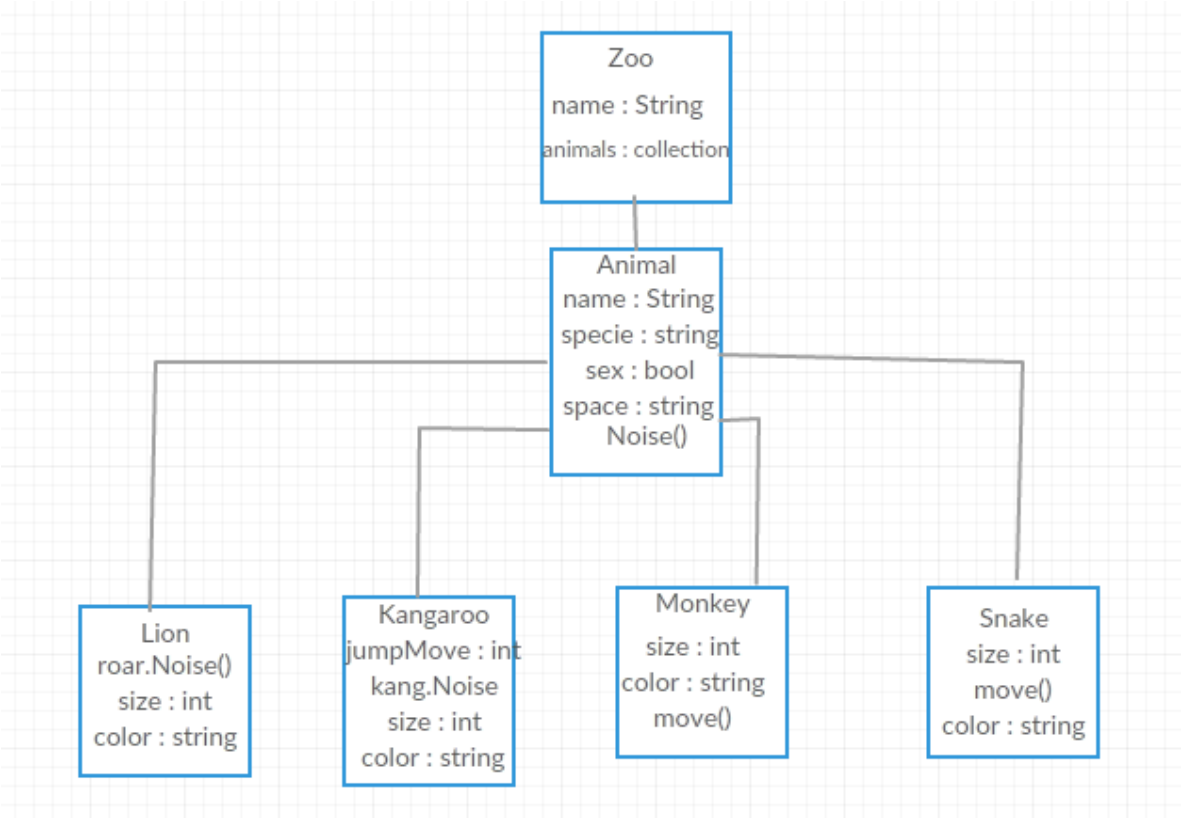
La herencia múltiple, de cara a la consistencia de los programas y los lenguajes tiene una relativamente alta complejidad. De ahí que algunos lenguajes orientados a objetos la permitan y otros no. Java no permite la herencia múltiple, pero a cambio dispone de la

construcción denominada “Interface” que permite una forma de simulación o implementación limitada de la herencia múltiple.

2.

a. a. Zoo – Diseño Top-Down

Usted ha sido contactado por un zoológico para diseñar un simulador que sirva para estudiar el reino animal. Usted debe crear un diagrama de clases que represente los animales del zoológico. Cada animal (EJ, León, Perro, Zorro, Lobo, Hipopótamo, Pez, Gato, Delfín, Águila) va a ser representado como un objeto. Use todos los atributos y métodos que considere relevantes para el modelado.



b. Figuras

```
package figure;

public class TestShapes {

    public static void main(String[] args) {

        Rectangle Uno = new Rectangle();

        Uno.setLenght(3);

        Uno.setWidth(7);

        System.out.println("Rectangulo 1"+"\nÁrea:"+Uno.recarea()+"\nPerímetro:
"+Uno.perimeter());
```

```
        TriangleRectangle Dos = new TriangleRectangle();
        Dos.setBase(5);
        Dos.setHeight(4);
        System.out.println("Triangulo Rectangulo
2"+"\nÁrea:"+Dos.triarea()+"\nPerímetro: "+Dos.triperimeter());


        TriangleIsos Tres = new TriangleIsos();
        Tres.setBase(10);
        Tres.setHeight(2);
        System.out.println("Triangulo Isosceles
3"+"\nÁrea:"+Tres.triarea()+"\nPerímetro: "+Tres.isoperimeter());


        Paralelogram Cuatro = new Paralelogram();
        Cuatro.setother(8);
        Tres.setHeight(1);
        System.out.println("Triangulo Isosceles
3"+"\nÁrea:"+Cuatro.parea()+"\nPerímetro: "+Cuatro.pperimeter());




}
}
//SUPERCLASE PARA TRIANGULOS
public class SuperTriangle {
    public double iBase = 1;
    public double iHeight = 1;
    public double getBase(){
        return this.iBase;
      }
      public double getHeight(){
        return this.iHeight;
      }
```

```
        public void setBase(double Base){
        this.iBase = Base;
               }

      public void setHeight(double Height){
        this.iHeight = Height;
}
        public double triarea(){
            return ((this.iBase * this.iHeight)/2);
      }



}
//SUPERCLASE PARA RECTANGULOS

    public class Rectangle extends SuperCuad{


      public double perimeter(){
             return super.perimeter(this.iLenght, this.iWidth);
      }
        public double recarea(){
             return super.area(this.iLenght, this.iWidth);
      }
       }

//FIGURAS
    public class TriangleRectangle extends SuperTriangle{


        public double triperimeter(){
```

```
            double h;
            h = Math.sqrt(Math.pow(this.iBase, 2)+Math.pow(this.iHeight, 2));
            return this.iBase + this.iHeight + h;
       }



}
public class TriangleIsos extends SuperTriangle{

        public double isoperimeter(){
            double h;
            h = Math.sqrt(Math.pow(this.iBase, 2)+Math.pow(this.iHeight, 2));
            return this.iBase + 2*h;
       }

}
public class Paralelogram extends SuperCuad{
    public double other = 1;
    //lado del paralelogramo

     public void setother(double Other){
            this.other = Other;
        }
        public double getother(){
            return other;
        }
    public double pperimeter(){
           return super.perimeter(this.other, this.iWidth);
      }
    public double parea(){
           return this.other*this.iWidth;
```

```
	}
}


    public class Rectangle extends SuperCuad{



	public double perimeter(){
		return super.perimeter(this.iLenght, this.iWidth);
	}
	  public double recarea(){
		return super.area(this.iLenght, this.iWidth);
	}
	 }
```

Hay similaridades entre las clases rectangulares y triangulares, debido a su parecido geometrico. Asi, las clases generales podrían ser Rectangle y Triangulo, asi como el main para testear los atributos.

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

La generalización parece ser una mejor alternativa cuando se trabaja con muchas características similares entre objetos, asi llegar a un nivel mas alto de clases ahorrando trabajo en código.

La especialización se ve como trabajar desde cero teniendo claras las características mas notorias en el nivel más alto, e ir creando atributos en los niveles mas bajos

Referencias

- Introduction to Java Programming, Brief, 8/E. Y. Daniel Liang, *Armstrong State University*
- Java Course online (https://staff.science.uva.nl/a.j.p.heck/Courses/JAVAcourse/ch3/s1.html)
- Mundojava.net - http://www.mundojava.net
- Aprendeaprogramar - http://aprenderaprogramar.com/