**TALLER 3**

**RELACIONES ENTRE OBJETOS**

**Integrantes**

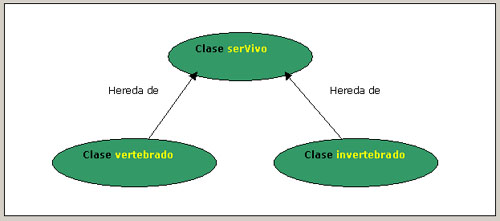
* Natalia Grijalba [ngrijalbah@unal.edu.co](mailto:ngrijalbah@unal.edu.co)
* David Fajardo [dsfajardob@unal.edu.co](mailto:dsfajardob@unal.edu.co)
* Cristian Bernal [crabernalmo@unal.edu.co](mailto:crabernalmo@unal.edu.co)
* Daniel Osuna [dgosunar@unal.edu.co](mailto:dgosunar@unal.edu.co)

1. Responda las siguientes preguntas:
2. **¿Cuál cree que es rol de herencia en un programa de Java?**
3. **¿Cómo la herencia promueve la reutilización de software?**

La herencia es una forma de reutilización de software en la cual, las nuevas clases se crean a partir de clases existentes al absorber sus atributos y comportamientos, y se mejoran con nuevas capacidades, o con modificaciones en las capacidades ya existentes. La herencia promueve la reutilización de software comprobado, depurado y de alta calidad, con lo que reduce los problemas una vez que el sistema se hace funcional.

1. **¿Cómo se podría explicar la Jerarquía (Hierarchy) en la Programación Orientada a Objetos?**

La Jerarquía la podríamos explicar de la siguiente manera: es la propiedad que nos permite la ordenación de las abstracciones.



1. **Explique la diferencia entre composición y herencia. De un ejemplo.**

La diferencia entre composición y herencia es la siguiente:

**HERENCIA:** permite definir una clase a partir de otra.

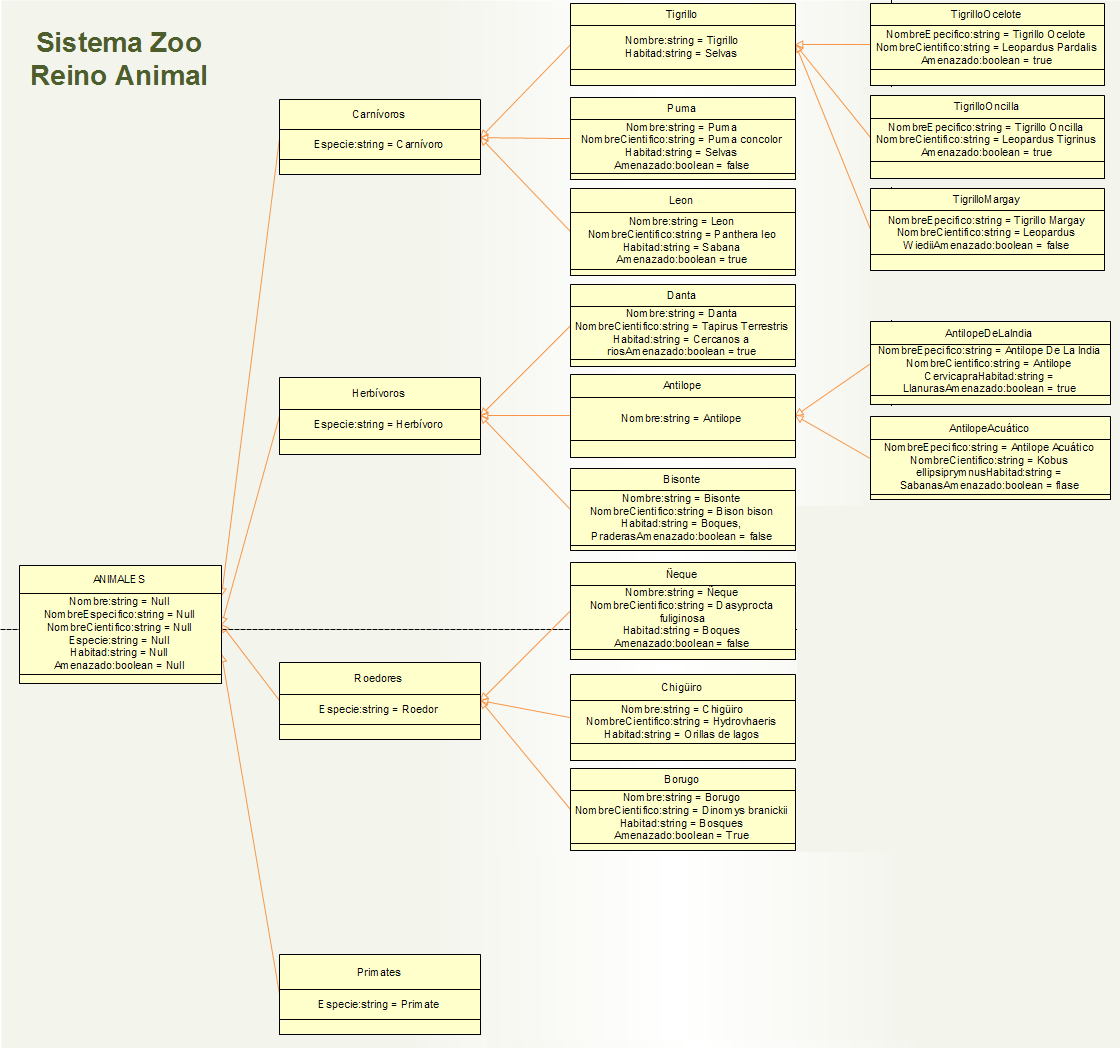
* Reutilización de caja blanca: los aspectos internos de la superclase son visibles a las subclases.
* Soportada por el lenguaje de programación.
* Estática: se define en tiempo de compilación.

**COMPOSICION:** nueva funcionalidad mediante composición de objetos.

* Reutilización de caja negra: no hay visibilidad de los aspectos internos de los objetos.
* Requiere interfaces bien definidas.
* Dinámica: se define en tiempo de ejecución.

**EJEMPLO:**

1. **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.**
2. Herencias:
3. **Zoo**

****

1. **Figuras**

Class Rectangle

Class Rectangle Triangle

Class Parallelogram

Class Isoceles Triangle

Length

Width

Arear

Perimeterr

Base

Heigth

Areart

Perimeterrt

Base

Heigth

Areait

Perimeterit

Length

Width

Areap

Perimeterp

Test Shapes

* ¿Cuáles características tienen en común?

Cada clase necesita de ingresársele dos valores, esta toma los valores y la figura que se está evaluando y calcula el área y perímetro de esta.

* ¿Cómo se crean las clases más generales?

Cada clase se crea con un nombre específico, se creó una clase para cada figura y en esta se definieron las variables seleccionadas para analizar la figura.

1. Mas Figuras:

**CLASE 1 “point”:**

Dentro de esta clase definimos el comportamiento para la obtención de los diferentes puntos en el plano de nuestras figuras geométricas, y a su vez encontramos el lado más largo y el más corto de alto y ancho para cada figura:

**import** java.util.\*;

**public** **class** point {

/\*definimos dos arreglos para los puntos en "x" y en "y"

\* junto con las variables correspondientes a los lados mayores y menores

\*/

**double** [] points\_x= **new** **double** [4];

**double** [] points\_y= **new** **double**[4];

**double** max\_x=0;

**double** max\_y=0;

**double** min\_x=0;

**double** min\_y=0;

//Aqui tenemos el constructor de la clase punto y su comportamiento general

**public** point(**double** [] point\_x, **double** [] point\_y){

**this**.points\_x= point\_x;

**this**.points\_y= point\_y;

Scanner sc = **new** Scanner(System.***in***);

//este for se utiliza para capturar los puntos "x" y "y" en arreglos

**for**(**int** d=0; d<2 ;d++ ){

System.***out***.println("please type x for the top");

**double** x=0;

x= sc.nextDouble();

points\_x[d]=x;

System.***out***.println("please type y for the top");

**double** y=0;

y= sc.nextDouble();

points\_y[d]=y;

}

**for**(**int** d=2; d<4 ;d++ ){

System.***out***.println("please type x for the bot");

**double** x= sc.nextDouble();

points\_x[d]=x;

System.***out***.println("please type y for the bot");

**double** y= sc.nextDouble();

points\_y[d]=y;

}

//Aqui asigamos el valor para cada lado de la figura

**double** width\_bot = Math.*abs*(max\_x= points\_x[1])+

Math.*abs*(max\_x= points\_x[0]);

**double** width\_top = Math.*abs*(max\_x= points\_x[2])+

Math.*abs*(max\_x= points\_x[3]);

**double** height\_bot = Math.*abs*(max\_y= points\_y[1])+

Math.*abs*(max\_y= points\_y[0]);

**double** height\_top = Math.*abs*(max\_y= points\_y[2])+

Math.*abs*(max\_y= points\_y[3]);

//con estos if encontramos que lado es mayor y que lado es menor

**if**(width\_bot<width\_top){

max\_x= width\_top;

min\_x= width\_bot;

}

**else**{

max\_x= width\_bot;

min\_x= width\_top;

}

**if**(height\_bot<height\_top){

max\_y= height\_top;

min\_y= height\_bot;

}

**else**{

max\_y= height\_bot;

min\_y= height\_top;

}

sc.close();

}

}

* CLASE 2 “Quiadrilateral”:

En esta clase tenemos la lógica que le conrrespondera a cada figura, aquí aplicaremos la clase point y daremos los mensajes correpondientes para cada coordenada en el plano, y de paso definimos el método del área:

**import** java.util.\*;

**public** **class** Quiadrilateral {

Scanner sc = **new** Scanner(System.***in***);

**double** height=0;

**double** widht=0;

String message = "Coordinates of Quadrilateral are:";

//Este constructor es el que nos mostrara el mensaje de coordenadas más adelante

**public** Quiadrilateral(String message){

**this**.message=message;

}

//Aqui Encontramos y definimos las coordenadas para cada punto

**public** **final** String points(){

**double** [] x = **new** **double**[4];

**double** [] y = **new** **double**[4];

point pt = **new** point(x,y);

String c1 ="(";

String c2 =")";

String c3 =",";

String p1= c1+x[0]+c3+y[0]+c2;

String p2= c1+x[1]+c3+y[1]+c2;

String p3= c1+x[2]+c3+y[2]+c2;

String p4= c1+x[3]+c3+y[3]+c2;

System.***out***.println(message);

//definimos alto y ancho de cada cuadrilatero

**double** height=pt.max\_y;

**double** widht=pt.max\_x;

**return** p1+p2+p3+p4;

}

//Simplemente damos una formula general para el area de los cuadrilateros

**public** **double** Area(){

**return** (height\*widht);

}

}

* CLASE 3 “Square”:

Desde esta clase cada figura tiene un comportamiento claro y predefinido por la clase “Quiadrilateral” (a excepción de dos figuras donde más adelante aclararemos su discrepancia frente al resto)

**public** **class** Square **extends** Quiadrilateral{

//Publicamos el mensaje de coordenadas

**public** Square(String message) {

**super**(message);

}

//hacemos el constructor para el area y retornamos el metodo inicial de super clase

**public** **double** Area() {

**return** **super**.Area();

}

}

* CALSE 4 “Rectangle”:

Clase hija de “Square” con un comportamiento prácticamente idéntico a la anterior nombrada

**public** **class** Rectangle **extends** Square {

//Publicamos el mensaje de coordenadas

**public** Rectangle(String message) {

**super**(message);

}

//hacemos el constructor para el area y retornamos el metodo inicial de super clase

**public** **double** Area() {

**return** **super**.Area();

}

}

* CLASE 5 “Parallelogram”:

Clase hija de “Quiadrilateral” la única diferencia con la super clase recae en la sobre escritura del método Area() pues para este tipo de cuadriláteros su forma varia.

**public** **class** Parallelogram **extends** Rectangle {

//Publicamos el mensaje de coordenadas

**public** Parallelogram(String message) {

**super**(message);

}

//Reescribimos este metodo pues la manera de encontrar estos valores cambia drasticamente

**public** **double** Area() {

//hacemos alucion a los arreglos de puntos antes instanciados

**double** [] x = **new** **double**[4];

**double** [] y = **new** **double**[4];

//Definimos es tamaño de cada lado de la figura

**double** top\_base= x[2]+x[3];

**double** top\_heigth= y[2]+y[0];

**double** bot\_base= x[0]+x[1];

**double** bot\_heigth= y[1]+y[3];

//Aplicamos la formula de area para cada triangulo obtenido

**double** area1= ((top\_base\*top\_heigth)/2);

**double** area2= ((bot\_base\*bot\_heigth)/2);

//retornamos el area final de nuestra figura

**return** area1+area2;

}

}

* CLASE 6 “Trapezoid”:

Clase hija de “Parallelogram” con un comportamiento prácticamente idéntico a la anterior nombrada

**public** **class** Trapezoid **extends** Parallelogram {

//Publicamos el mensaje de coordenadas

**public** Trapezoid(String message) {

**super**(message);

}

//hallamos el area que es la misma que la de su clase padre

**public** **double** Area() {

**return** **super**.Area();

}

}