



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.C. Jorge Rommel Santiago Arce

Asignatura: Programación Orientada a Objetos

Grupo: 2

No de Práctica(s): 7

Integrante(s): Cruz Rangel Leonardo Said
Ibañez Guzman Osvaldo
Maya Ortega Maria Fernanda
van der Werff Vargas Pieter Alexander

*No. de Equipo de
cómputo empleado:* 45-48

No. de Lista o Brigada: 7

Semestre: 2020-1

Fecha de entrega: 30/09/2019

Observaciones:

CALIFICACIÓN: _____

PRÁCTICA 7. “Herencia”

Objetivos:

Implementar los conceptos de herencia en un lenguaje de programación orientado a objetos.

Actividad 1 - Crear clases que implementen herencia:

En la primera parte de esta sección se genera un error en las variables “x” y “y” ya que como lo indica la consola, son de acceso privado, por lo que se intenta acceder a los atributos desde ambas clases y genera un error.

```
[Granada47:Equipo7_Practica7 poo02alu37$ javac Punto.java Circulo.java
Circulo.java:10: error: x has private access in Punto
    x = valorX;
    ^
Circulo.java:11: error: y has private access in Punto
    y = valorY;
    ^
Circulo.java:36: error: x has private access in Punto
    return "Centro = ["+x+", "+y+"]; Radio = "+radio;
                   ^
Circulo.java:36: error: y has private access in Punto
    return "Centro = ["+x+", "+y+"]; Radio = "+radio;
                   ^
4 errors
```

Después de modificar los atributos para cambiarlos a protected, cada clase puede ocupar sus propios atributos “x” y “y”, de manera que la clase Punto sea la única que acceda a esos atributos.

```
[Granada47:Equipo7_Practica7 poo02alu37$ javac Punto.java Circulo.java
```

Actividad 2 -

Otra manera en la que se puede modificar a las clases Punto y Círculo, de manera que no genere errores y puedan funcionar las clases; es agregando un llamado a un método super dentro de la clase Círculo para acceder a las variables dentro de la clase Punto. De esta manera, se puede regresar en la clase Punto a los atributos a private en lugar de protected.

```
[Granada47:Equipo7_Practica7 poo02alu37$ javac Punto.java Circulo.java
```

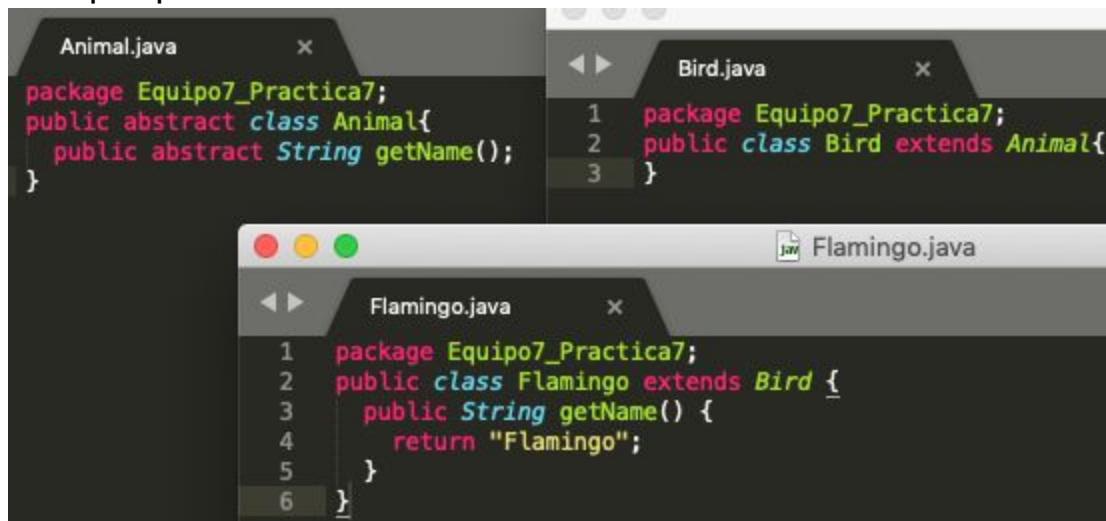
Clase cilindro y cliente de la clase cilindro.

La clase cilindro va a heredar los métodos de la clase Círculo y además va a agregar un atributo altura, como Círculo tiene un constructor es necesario definir uno a nuestra clase hija, el cual modificará los valores de x,y y radio de la clase padre por medio del uso de super y además modificará el atributo altura. La clase cilindro también contendrá el método accesor y mutador del atributo altura y contará con un accesor del área.

```
C:\Users\essmo\OneDrive\Escritorio\Equipo7_Practica7>javac *.java
C:\Users\essmo\OneDrive\Escritorio\Equipo7_Practica7>java -classpath C:\Users\essmo\OneDrive\Escritorio Equipo7_Practica7.Cilindro
7.Cilindro
849.4866535306801
```

Actividad 3 -

Al momento de hacer la herencia entre las clases concretas es necesario que la primera clase que herede use en la totalidad los métodos abstractos, por lo que en la primera parte del ejercicio no logra compilar los archivos porque la clase Bird no usa los métodos de Animal.



```
Animal.java
package Equipo7_Practica7;
public abstract class Animal{
    public abstract String getName();
}

Bird.java
1 package Equipo7_Practica7;
2 public class Bird extends Animal{
3 }

Flamingo.java
1 package Equipo7_Practica7;
2 public class Flamingo extends Bird {
3     public String getName() {
4         return "Flamingo";
5     }
6 }
```

```
Georgia46:Equipo7-Practica7 poo02alu19$ javac Animal.java Bird.java Flamingo.java
Bird.java:2: error: Bird is not abstract and does not override abstract method getName() in Animal
public class Bird extends Animal{
      ^
1 error
Georgia46:Equipo7-Practica7 poo02alu19$
```

Para el ejercicio 2 se cambiaron las herencias de Bird ahora hereda de Flamingo y Flamingo de Animal, esto ya está permitido ya que la primera clase que hereda de Animal usa en su totalidad los métodos abstractos, sin embargo esto no manda nada a pantalla ya que no se está utilizando algún método main.

```
package Equipo7_Practica7;
public abstract class Animal{
    public abstract String getName();
}

1 package Equipo7_Practica7;
2 public class Bird extends Flamingo{
3 }

Flamingo.java
1 package Equipo7_Practica7;
2 public class Flamingo extends Animal {
3     public String getName() {
4         return "Flamingo";
5     }
6 }

[Georgia46:Equipo7_Practica7 poo02alu19$ javac Animal.java Bird.java Flamingo.java
[Georgia46:Equipo7_Practica7 poo02alu19$ java Animal
Error: no se ha encontrado o cargado la clase principal Animal
```

De igual forma esto funciona si todas las clases están en un mismo archivo, solo que se deben de omitir la palabra “public” para que pueda compilarse.

```
package Equipo7_Practica7;
public abstract class Animal{
    public abstract String getName();
}

class Flamingo extends Animal {
    public String getName() {
        return "Flamingo";
    }
}

class Bird extends Flamingo{
}

[Georgia46:Equipo7_Practica7 poo02alu19$ javac Animal.java
[Georgia46:Equipo7_Practica7 poo02alu19$ ]
```

Conclusiones:

Cruz Rangel Leonardo Said:

La herencia en java es bastante importante dentro de la programación, pues nos permite reutilizar código al momento de querer generar objetos con las mismas características que su padre y agregarle algunas otras.

Ibañez Guzman Osvaldo:

En esta práctica se vio la importancia de algunos comandos en java, además de la importancia que tiene la herencia en la programación para poder hacer más fácil la implementación de código.

Maya Ortega María Fernanda:

Como podemos recordar de las clases teóricas una de las características más importantes en la programación orientada a objetos es la herencia, junto con el manejo de superclases y subclases para poder modificar los métodos de las primeras y ocuparlos de manera eficaz en las segundas.

van der Werff Vargas Pieter Alexander:

El concepto de herencia es útil cuando se desea ocupar el código dentro de una clase ya previamente definida, pero puede que se quiera agregar o modificar alguna parte, sin alterar el código anterior. Por lo que es necesario saber cómo llamar a un atributo o método de la otra clase mediante súper, así como el orden de compilación de las clases y su tipo de implementación.