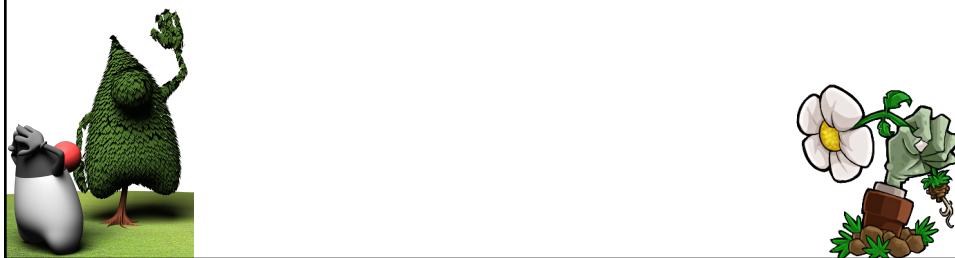


Pilares de la Programación Orientada a Objetos

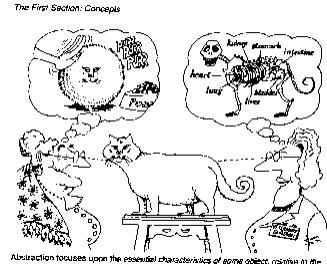


2

Principios de la POO

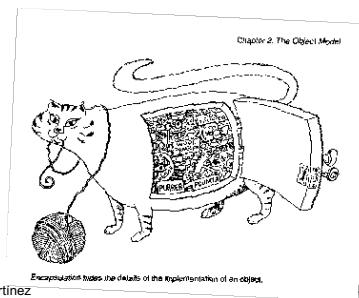
□ Abstracción

- Descripción simplificada de un sistema que enfatiza algunas propiedades y suprime otras
- Permite definir nuevos tipos de datos por el usuario



□ Encapsulación

- Permite ocultar información, que no tiene porque ser usada fuera de la clase



Carmen Villar / René Martínez



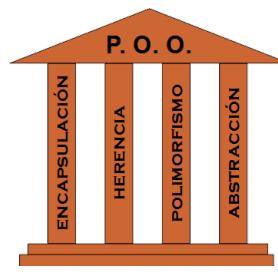
□ Herencia

- Designa la facultad de los objetos de compartir propiedades y operaciones entre ellos
- Permite crear nuevas clases a partir de clases ya existentes

□ Polimorfismo

- Permite aplicar una misma operación sobre objetos de diferentes clases

Carmen Villar / René Martínez



6

6

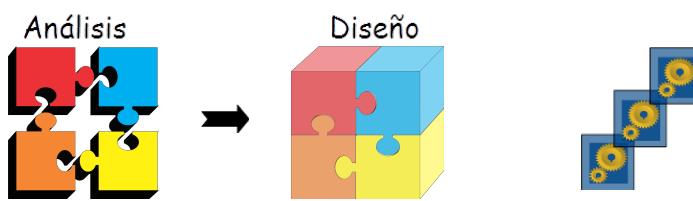
Análisis y diseño

□ Análisis

- Se especifica **qué** debe hacer el sistema
- Modela las actividades, objetos y comportamientos

□ Diseño

- Se describe **cómo** lo va a hacer
- Modela las relaciones entre los objetos y encuentra abstracciones útiles que simplifiquen el problema



Carmen Villar / René Martínez

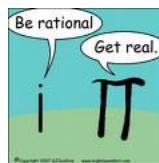
8

8

Abstracción

□ Abstracción

- Modelo que resalta las características de interés
- Agrupa un conjunto de atributos y comportamientos en una clase
- Permite crear conjuntos de objetos que soportan una actividad compleja



Encapsulación



- Agrupa datos y sus operaciones
- Controla el acceso a los datos internos
- Hace el código más fácil de mantener

```
MyDate
+day : int
+month : int
+year : int
```

```
MyDate d = new MyDate();
d.day = 32;
// invalid day

d.month = 2; d.day = 30;
// plausible but wrong

d.day = d.day + 1;
// no check for wrap around
```

```
MyDate
-day : int
-month : int
-year : int

+getDay() : int
+getMonth() : int
+getYear() : int
+setDay(int) : boolean
+setMonth(int) : boolean
+setYear(int) : boolean
```

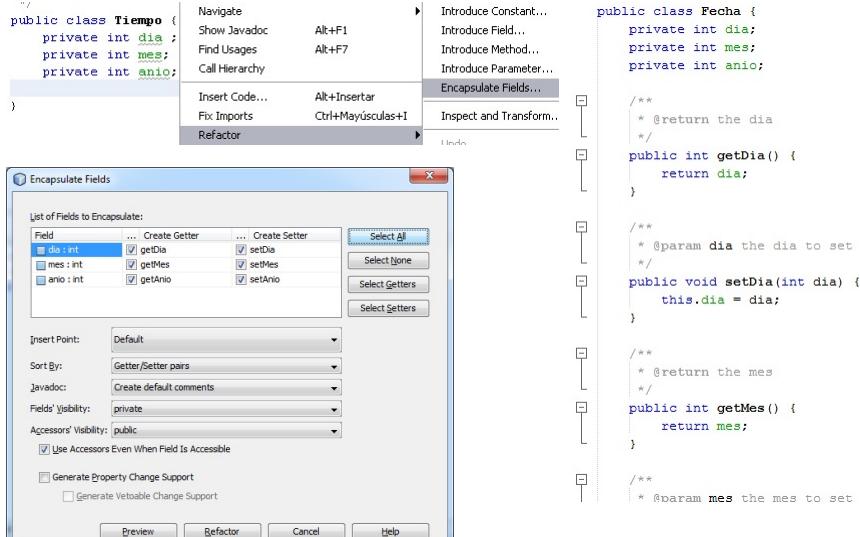
Verify days in month

```
MyDate d = new MyDate();
d.setDay(32);
// invalid day, returns false

d.setMonth(2);
d.setDay(30);
// plausible but wrong,
// setDay returns false

d.setDay(d.getDay() + 1);
// this will return false if wrap around
// needs to occur
```

Netbeans y la encapsulación



Carmen Villar / René Martínez

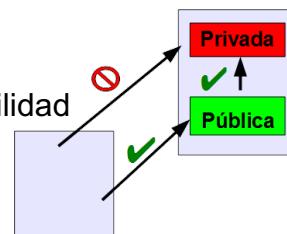
14

14

Encapsulación: Permisos

□ Oculta información

- Datos y métodos
- Se controla con atributos de Visibilidad



Atributo	M i e m b r o d e :			
	Misma clase	Clase derivada	Mismo paquete	Otro paquete
private	x			
protected	x	x	x	
default	x		x	
public	x	x	x	x

Carmen Villar / René Martínez

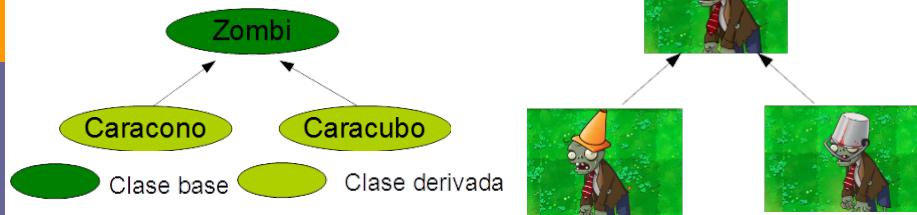
16

16

Herencia

□ Permite diseñar dos entidades diferentes (clases) pero que tienen varias características comunes

- Extiende la funcionalidad de un objeto
- Crear nuevos tipos (clases derivadas) con las propiedades extendidas del original (clase base)
- Ayuda a la reutilización de código

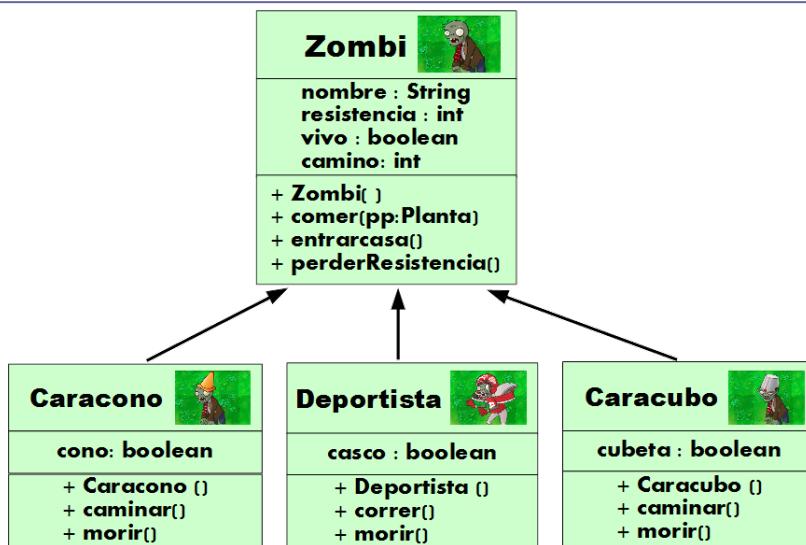


Carmen Villar / René Martínez

18

18

Ejemplo Herencia

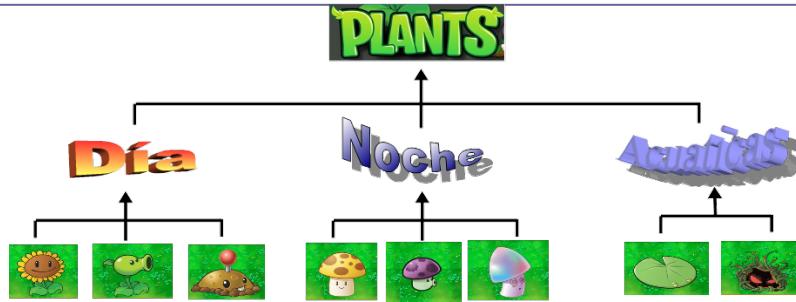


Carmen Villar / René Martínez

20

20

Jerarquías de herencia



```

java.lang.Object
  ↳ java.awt.Component
    ↳ java.awt.Container
      ↳ javax.swing.JComponent
        ↳ javax.swing.AbstractButton
          ↳ javax.swing.JButton
  
```

Carmen Villar / René Martínez

22

22

Instrucciones para herencia en Java

- La clase derivada hereda los atributos y métodos de la clase base
 - Excepto los atributos por defecto y los privados, así como los métodos estáticos

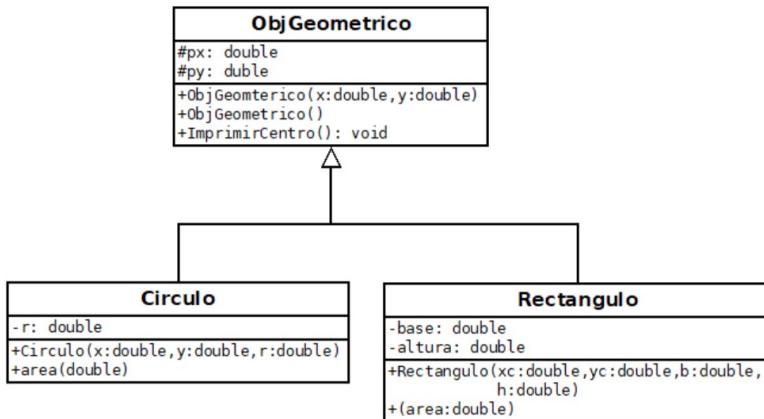
```
class Clase_Derivada extends Clase_Base
```
- Constructores
 - No se heredan
 - El constructor de la clase derivada, debe llamar primero al de la clase base
 - La instrucción super, invoca al constructor de la clase base


```
modificador ClaseDerivada(parametros){
    super(parametros constructor clase base);
    //instrucciones constructor clase derivada
}
```



24

Ejemplo usando UML



Carmen Villar / René Martínez

26

26

Clases:ObjGeometrico y Circulo

```

package figuras;
public class ObjGeometrico {
    protected double px, py;
    public ObjGeometrico(double x, double y){px=x; py=y;}
    public ObjGeometrico(){ px=py=0; }
    public void imprimirCentro()
        { System.out.println("(" +px+ "," +py+ ")" ); }
}
  
```

```

package figuras;
public class Circulo extends ObjGeometrico{
    private double r;
    public Circulo (double x, double y, double r) {
        super(x,y);
        this.r=r;
    }
    public double area() { return Math.PI * r * r; }
}
  
```

Carmen Villar / René Martínez

28

28

Clase: Rectangulo

```
package figuras;
public class Rectangulo extends ObjGeometrico {
    private double base, altura;
    public Rectangulo(double xc, double yc,
                      double b, double h){
        super(xc,yc);
        base= b;
        altura= h;
    }
    public double area(){
        return base*altura;
    }
}
```

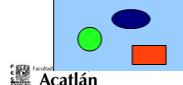


Ejemplo. Aplicación: PruebaFiguras

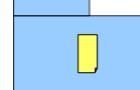
```
package herencia;
import figuras.*;
class PruebaFiguras {
    public static void main(String[] args) {
        Circulo cr = new Circulo(2.0,2.5,2.0);
        Rectangulo cd = new Rectangulo(3.0,3.5,4.0,3.0);
        System.out.print("Centro del circulo:");
        cr.imprimirCentro();
        System.out.print("Centro del rectangulo:");

        cd.imprimirCentro();
        System.out.println("Area del circulo:"+cr.area());
        System.out.println("Area del rectangulo:"+cd.area());
    }
}
```

figuras



herencia



```
run:
Centro del circulo:(2.0,2.5)
Centro del rectangulo:(3.0,3.5)
Area del circulo:12.566370614359172
Area del rectangulo:12.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejercicio

- Crea el paquete figuras
 - Coloca las clases de ObjGeometrico, Circulo y Rectangulo
- Crea el paquete herencia
 - Coloca el Java Main Class de PruebaFiguras
- Programa la clase Elipse e inclúyela al paquete figuras
 - Debe heredar de ObjGeometrico
 - Recibe los valores de a (semieje menor) y b (semieje mayor)
 - El area= PI*a*b.

Ejercicio

- Modifica PruebaFiguras para verificar que tu clase funciona
 - Centrada en (5,5) y con semieje menor de 2 y mayor de 4

Operador instanceof

- Permite reconocer la clase a la que pertenece un objeto

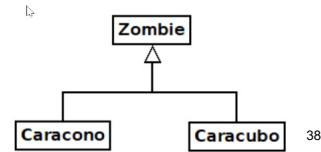
```
nombre_objeto instanceof
Nombre_clase
z0 instanceof Zombie
z1 instanceof Zombie
z2 instanceof Zombie
z3 instanceof Zombie
z1 instanceof Caracono
z2 instanceof Caracono
z3 instanceof Caracono
z1 instanceof Caracubo
z3 instanceof Caracubo
```

Regresa true o false

Ejemplo:

```
Zombie z0 = null;
Zombie z1 = new Zombie();
Caracono z2 = new Caracono();
Caracubo z3 = new Caracubo();
```

Carmen Villar / René Martínez



38

38

Definición Polimorfismo

- El polimorfismo se usa en jerarquías de herencia
- La variable de una superclase puede recibir el mismo mensaje para diferentes subclases
- Capacidad de referirse a un objeto a través de una referencia del tipo de su superclase

Definición Polimorfismo (cont)

□ Ejemplo:

```
ObjGeometrico fig;
fig = new Circulo(2.0,2.5,2.0);
System.out.print("Centro del circulo:");
fig.imprimirCentro();
fig = new Rectangulo(3.0,3.5,4.37,3.85);
System.out.print("Centro del rectangulo:");
fig.imprimirCentro();
```

□ El método `imprimirCentro()`, es un método polimórfico



Ejemplo polimorfismo

```
package herencia;
import figuritas.*;
public class PolimorFiguras {
    public static void main(String[] args) {
        ObjGeometrico fig;
        fig = new Circulo(2.0,2.5,2.0);
        System.out.print("Centro del circulo:");
        fig.imprimirCentro();
        System.out.println("Area del circulo:"+
                           ((Circulo)fig).area());
        fig = new Rectangulo(3.0,3.5,4.37,3.85);
        System.out.print("Centro del rectangulo:");

        fig.imprimirCentro();
        System.out.println("Area del rectangulo:"+
                           ((Rectangulo)fig).area());
```

run:
Centro del circulo:(2.0,2.5)
Area del circulo:12.566370614359172
Centro del rectangulo:(3.0,3.5)
Area del rectangulo:16.8245
BUILD SUCCESSFUL (total time: 0 seconds)



Utilidad del Polimorfismo

■ Ejemplo inspirado en Plantas .vs. Zombies

- Tenemos diferentes tipos de zombis
- ¿Cómo controlas a todos los zombies activos en el juego?

```
Zombie z0 = new Zombie();
Zombie z1 = new Zombie();
:
Zombie zn = new Zombie();
Caracono c0 = new Caracono();
:
Caracono cn = new Caracono();
Saltador s0 = new Saltador();
:
Saltador sn = new Saltador();
```

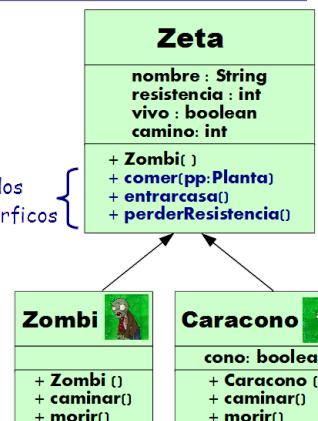


```
Zombie [] z = new Zombie[n];
Caracono [] c = new Caracono[n];
Saltador [] s = new Saltador[n];
z[i] = new Zombie();
c[i] = new Caracono();
s[i] = new Saltador();
```

Utilidad del Polimorfismo (cont.)

```
Zeta [] z = new Zeta[n];
:
while(!terminaJuego()){
    //Generar de forma aleatoria
    r = Math.random();
    if(r < 0.5)
        z[i] = new Zombie();
    else if (r < 0.8)
        z[i] = new Caracono();
    else
        z[i] = new Saltador();
    :
    z[i].comer(girasol());
    :
    if(z[i] instanceof Caracono)
        ((Caracono)z[i]).caminar();
    else if ...
    i++;
}
```

Métodos Polimórficos {



Algo más de los objetos



□ Permite reaprovechar código



□ Recolector de basura “Garbage Collector”

- Java borra automáticamente (libera memoria) los objetos que no se usan
 - Idea tomada de lenguajes como Lisp
- Proceso (hilo) de baja prioridad
 - Se ejecuta en tiempos de ocio

