

Programación 2



**Tecnicatura en Desarrollo de Aplicaciones
Informáticas**

Resumen

- Los Objetos encapsulan **Comportamiento** y **Estado**

El objeto **NO** es solo una entidad que contiene datos

Datos y comportamiento **fuertemente relacionados**

Resumen

- Los Objetos tienen **responsabilidades** y colaboran mediante el **envió de mensajes**

Cada objeto tiene una responsabilidad determinada, entre varios objetos logran alcanzar un objetivo

Resumen

- El **procesamiento** es realizado por los **objetos** que se comunican entre ellos

A partir del envío de mensajes los objetos realizan el procesamiento en conjunto (cada uno con su responsabilidad)

Resumen

- Cada **Objeto** tiene su propio **espacio de memoria**

Cada objeto ocupa un espacio en memoria, se crea y destruye en forma dinámica.

Resumen

- Un **objeto** es **instancia** de una **clase**
- Una **clase** define **objetos** similares

Resumen

- Una **clase** es un **molde** que define las **instancias**
- Todos los objetos que son instancia de la **misma clase** pueden realizar las **mismas operaciones**

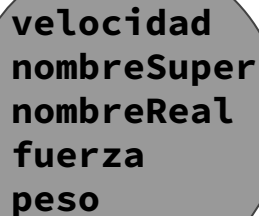
Definiciones

Atributo y variable de instancia

Un atributo especifica una cualidad de un objeto.

Una variable de instancia especifica cómo se almacenan los valores para esa cualidad.

El estado es el conjunto de valores que toman los atributos de un objeto en un determinado instante.




velocidad
nombreSuper
nombreReal
fuerza
peso



Atributo variable de instancia

Información o estado asociado con un componente



velocidad
nombreSuper
nombreReal
fuerza
peso



Clase

Un molde de objetos.

Una fábrica para instanciar objetos.

La descripción de una colección de objetos relacionados

CLASE



Instancia

Un objeto creado por una clase

Instancias de una misma clase



Instanciación

El acto de crear una
instancia

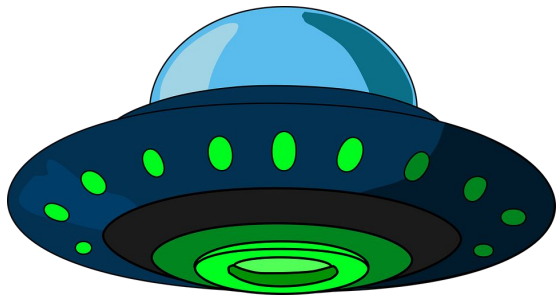


Objeto

Un componente de software.

Instancia de una clase.

Mínima unidad computacional que
encapsula **estado** y
comportamiento.



Mensaje

Un pedido enviado a un **objeto** que desencadena la ejecución de un **método**



Método

La **implementación** de
una operación



Composición

La construcción de un componente mediante otros Componentes.

Los **objetos** pueden contener otros **objetos**



Encapsulamiento

Los **datos** en los objetos son **privados**

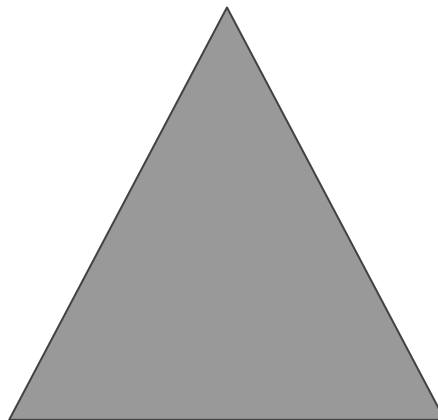
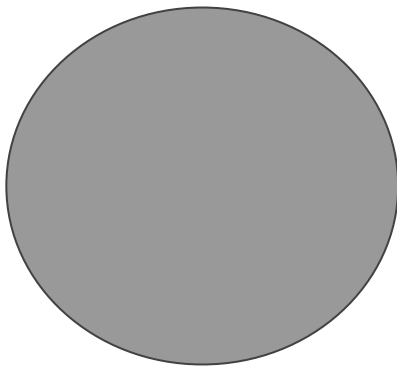
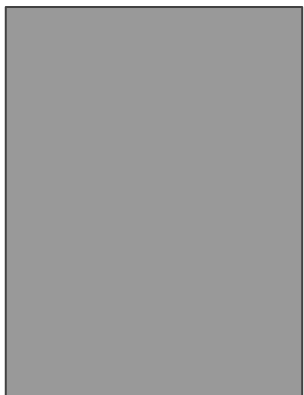
Los **métodos** son (típicamente) **públicos**



Figuras Geométricas

Figuras Geométricas

— — —



Herencia

Herencia

— — —

Mecanismo de abstracción, clasificación, extensión y reuso

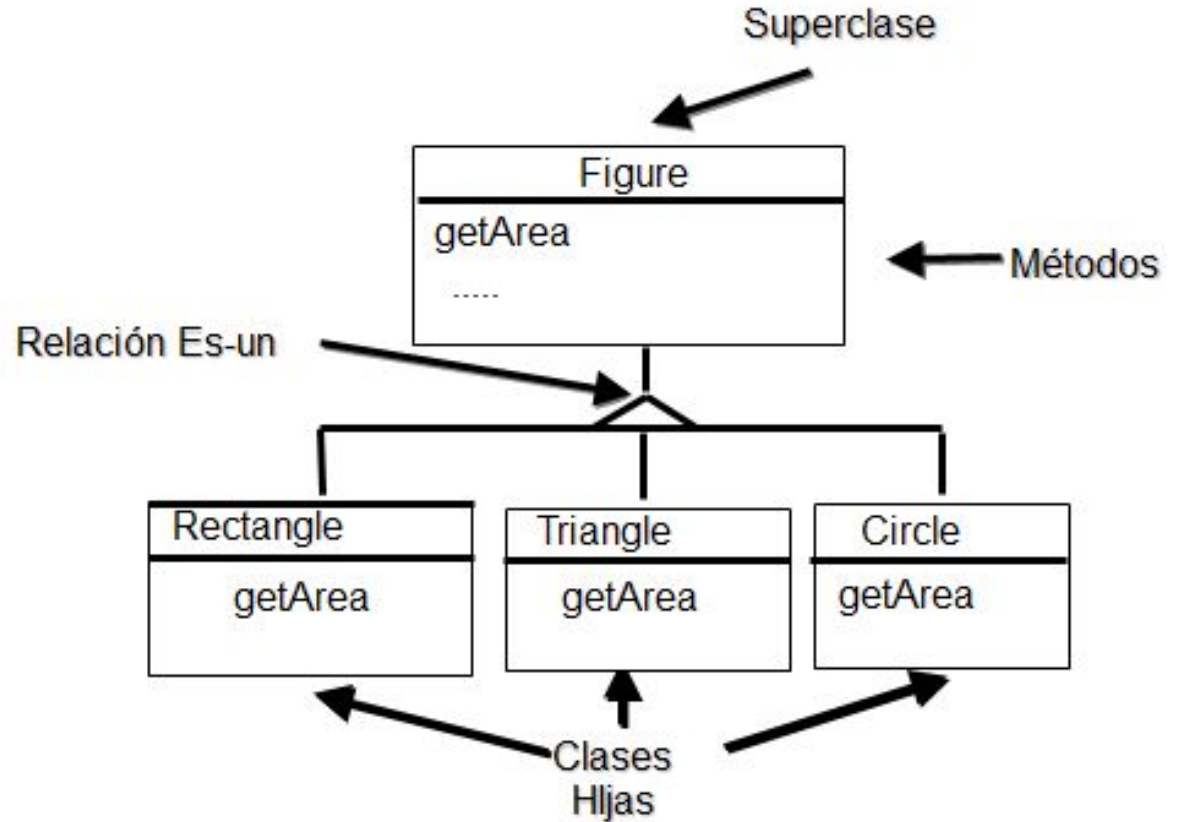
- Es posible abstraer características comunes de varias clases en una “superclase”
- EL mecanismo de abstracción sirve como mecanismo de clasificación de entidades
- La extensión permite ampliar las características de una clase en una subclase
- Es un mecanismo de reuso tanto a nivel de diseño como implementación

Que tienen en común Triangulo, Circulo y Rectangulo?

— — —

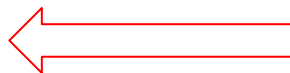


Herencia: Ejemplo



Herencia: Ejemplo

```
public class Figura {  
    String nombre;  
  
    public double getArea(){  
        return 0.0;  
    }  
}
```



Existe una simplificación del concepto para introducir la herencia más adelante en el curso vamos a ver el tema de métodos que “no hacen nada”

```
    public String getNombre(){ return nombre;}  
}
```

Herencia: Ejemplo

```
public class Circulo extends Figura{  
    double radio;  
  
    public double getArea(){  
        return Math.PI * radio * radio;  
    }  
}
```

Círculo Hereda de
Figura

El Círculo es una
Figura

Herencia: Ejemplo

```
public class Triangulo extends Figura{  
    double base;  
  
    double altura  
  
    public double getArea(){  
        return (base*altura)/2;  
    }  
}
```

Triangulo Hereda de
Figura

El Triángulo es una
Figura

Herencia Ejemplo Constructores

Los constructores no se heredan!!!

En la clase Figura:

```
public Figura(String n) {  
    nombre = n;  
}
```

Herencia Ejemplo Constructores

Los constructores no se heredan!!!

En la clase Circulo:

```
public Circulo(int r) {  
    nombre = "Circulo";  
    radio = r;  
}
```

```
public Circulo(int r) {  
    super("circulo");  
    radio = r;  
}
```

Se puede invocar el constructor de la clase padre (si o si primer línea)

Ejemplo de Uso de Herencia

```
Circulo c1 = new Circulo(4);
```

```
Triangulo t1 = new Triangulo(10,10);
```

```
c1 = t1;      // ERROR “un triángulo no es un círculo”
```

```
t1 = c1 ;    // ERROR “un círculo no es un triángulo”
```

```
Figura ff1 = t1; // SI ! “El triángulo es una figura”
```

```
ff1 = c1; // SI ! “El círculo es una figura”
```

Ejemplo: Envío de mensajes

```
c1.getArea();
```

```
c1.getNombre();
```

```
c1.getRadio();
```

```
ff1 = c1;
```

```
ff1.getRadio(); // ERROR, Java es un lenguaje Tipado
```

JAVA: Compilación

Java controla el envío de los mensajes por el **TIPO** del objeto, es decir el **control es estático**.

En el ejemplo anterior `ff1` es del tipo `Figura`, y la clase `Figura` no tiene un método `getRadio()`

super

Similar a **this**, la palabra **super** se utiliza para referir al “padre” de la clase. Lo usamos para poder invocar un método y modificar su comportamiento.

Supongamos una clase MedioCirculo (es un círculo pero que tiene la mitad de area)



super

— — —

```
public class MedioCirculo extends Circulo{  
    public double getArea(){  
        return super.getArea()/2;  
    }  
}
```

El MedioCirculo es un Círculo cuya área es la mitad del Círculo.
Por ejemplo si cambio el getArea del Círculo, el MedioCirculo sigue
cumpliendo la propiedad de que su área es la mitad de la de Círculo

Binding Dinámico

Es un mecanismo a través del cual el método que se ejecuta en respuesta a un mensaje se determina dinámicamente dependiendo de la clase a la que pertenezca la instancia que recibió el mensaje.

Binding Dinámico: Ejemplo

```
Figura f4 = new Triangulo(10,10);
```

```
f4.getArea(); // SI fuera estatico, se ejecuta el de la  
“clase” y no el del Objeto Recién en t de ejecucion se sabe  
el metodo
```

Binding Dinámico

Siguiendo el ejemplo

```
if (EL USUARIO APRIETA "1" )
```

```
    f4 = new Circulo(4);
```

```
else
```

```
    f4 = new Triangulo(10,10);
```

```
f4.getArea(); // Que metodo se ejecuta?
```

Binding Dinámico

Recordar que Mensaje era distinto a Método.

El mensaje es la señal que se envía, y el método el código que se ejecuta como respuesta a la señal

Polimorfismo

Griego (muchas Formas)

Es la habilidad de una **variable** o **referencia** de tomar valores de diferentes **tipos**, lo que implica la respuesta a los mismos **mensajes**.

Polimorfismo Ejemplo

```
public void imprimirFigura ( Figura ff){  
    System.out.println( “la figura “ + ff.getNombre() +  
        “ tiene un Area de: “ + ff.getArea() );  
}
```

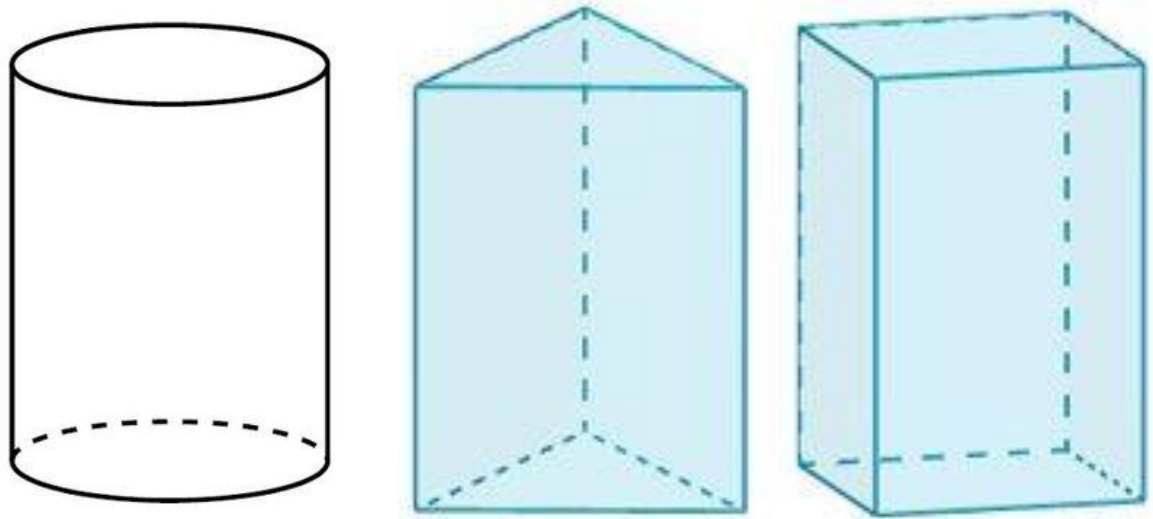
La variable ff puede tomar diversas “formas”, aunque siempre de las que hereden de Figura

**Polimorfismo y
binding dinámico
son dos mecanismos
esenciales que
permiten el reuso y son
la base de la potencia
y elegancia de la POO**



Ejemplo Figuras3D

Figura3D



Simplificación en la cual debemos calcular el volumen de la figura

Juego de Dados



Nueva Funcionalidad

Un jugador es tramposo y utiliza dados cargados que favorecen la salida del **5** y el **6** respectivamente la mitad de las veces que se tira el dado



Dados Cargados

```
public class Dado{  
    int valor;  
    int caras;  
  
    public int tirar() {  
        return (Math.radom()*caras)+1;  
    }  
}
```

```
public class DadoCargado5 extends Dado{  
  
    public int tirar() {  
        if (Math.random()>0.5)  
            return super.tirar();  
        else  
            return 5;  
    }  
}
```

```
public class DadoCargado6 extends Dado{  
  
    public int tirar() {  
        if (Math.random()>0.5)  
            return super.tirar();  
        else  
            return 6;  
    }  
}
```



Mal uso de la herencia: no voy a crear una clase por cada valor de una variable de instancia que cambie

Dados Cargados

```
public class Dado{  
    int valor;  
    int caras;  
  
    public int tirar() {  
        return (Math.radom()*caras)+1;  
    }  
}
```

```
public class DadoCargado extends Dado{  
  
    int ladoCargado;  
    public int tirar() {  
        if (Math.random()>0.5)  
            return super.tirar();  
        else  
            return ladoCargado;  
    }  
}
```

Dados Cargados

— — —

¿Qué otras clases debo modificar?

Como?

Clase Vs Instancia

No puede existir clases iguales

— — —

Dos clases iguales son la misma

CUANDO LA DIFERENCIA ES SOLO UNA CONSTANTE, entonces también son la misma clase

Malos ejemplos

PersonaJuan

PersonaPedro ----> “Juan”

----> Existe un nombre, las dos son personas con una variable Nombre

Mal en Dados

Clase padre Dado

DadoCargado5 y DadoCargado6 ---> Los dos son DadoCargado y el valor en realidad es una variable!!!

DESAPARECEN DADOCARGAD05 y DADOCARGAD06

Reconocer fácil el error

```
if (x>150)
```

NO TIENE QUE HABER
CONSTANTES EN EL
CÓDIGO

```
if (nombre.equals("juan"))
```

```
return "juan"
```

```
return 84;
```

```
return "a";
```

```
return "juan";  
reemplazar  
return nombreADevolver;
```

La Clase Object

Object

Object es una clase

La clase Object es la superclase de todas las clases

Todas las clases heredan directa o indirectamente de Object

Object

```
public class Object {  
  
    public boolean equals( Object obj );  
  
    public String toString( )  
  
    protected void finalize( )  
  
    ...}
```

Estos métodos
pueden ser
redefinidos en las
subclases si es
necesario

`equals()`

- Retorna true cuando dos objetos tienen iguales valores
- La implementación por defecto compara referencias con `==`

equals() Ejemplo Dado

```
public class Dado{  
    public boolean equals( Object obj ) { // PARA REDEFINIR  
// EL MÉTODO NO PUEDO CAMBIAR LA SIGNATURA DEL MISMO  
        Dado other = (Dado) obj; // SI NO ES UN DADO DA ERROR  
//MÁS ADELANTE VAMOS A VER COMO SE SOLUCIONA  
        return this.getValor() == other.getVALor();  
    }  
}
```

equals() Ejemplo Dado

— — —

```
Dado d1 = new Dado(3);
```

```
Dado d2 = new Dado(3);
```

```
(d1 == d2);      // returns false
```

```
d1.equals(d2);   // returns true
```

toString()

- `toString()` se usa para proveer una representación del objeto como una cadena de caracteres
- Invocada automáticamente cuando utilizamos
 - `System.out.println()` y `“+”`
 - `String s = “The object is “ + obj;`
 - `System.out.println(obj);`
- La implementación por defecto de la clase `Object` retorna el nombre de la clase y la ubicación en memoria del objeto

toString() Ejemplo

```
public class Dado {  
  
    public String toString( ) {  
        return "Dado con el valor = " + this.getValor();  
    }  
}
```

finalize()

Este método se llama justo antes de que un objeto sea basura recolectada (garbage collected).

Es invocado por el recolector de basura en un objeto cuando el recolector de basura determina que no hay más referencias al objeto.

Debemos usar el método `finalize()` para eliminar los recursos del sistema, realizar actividades de limpieza y minimizar las pérdidas de memoria