

# Programación 2



**Tecnatura 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 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

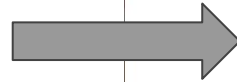


# Instancia

---

Un objeto creado por una clase

Instancias de una misma clase



# Instanciación

---

El acto de crear una  
instancia



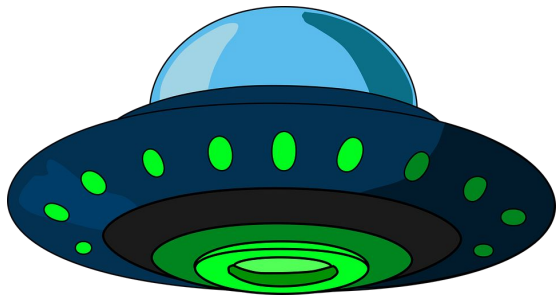
# Objeto

---

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**

**objeto.mensaje()**



# 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**

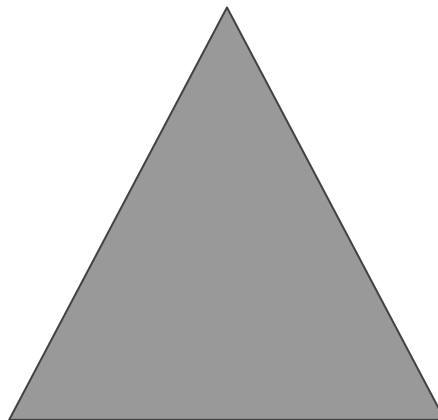
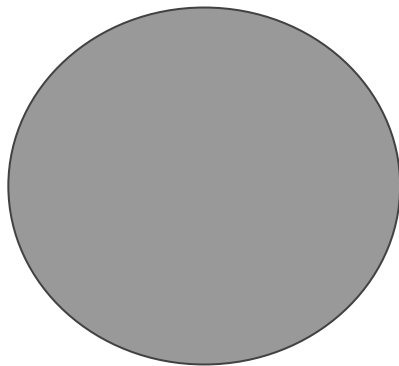
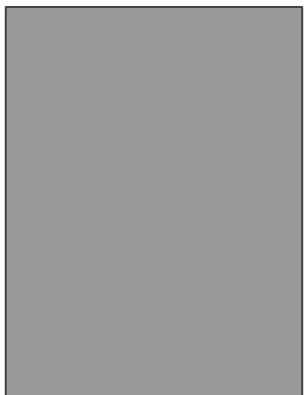
Los **métodos** son (típicamente) **públicos**



# Figuras Geométricas

# Figuras Geométricas

— — —



# Herencia

# Herencia

— — —

## Mecanismo de abstracción, clasificación, extensión y reuso

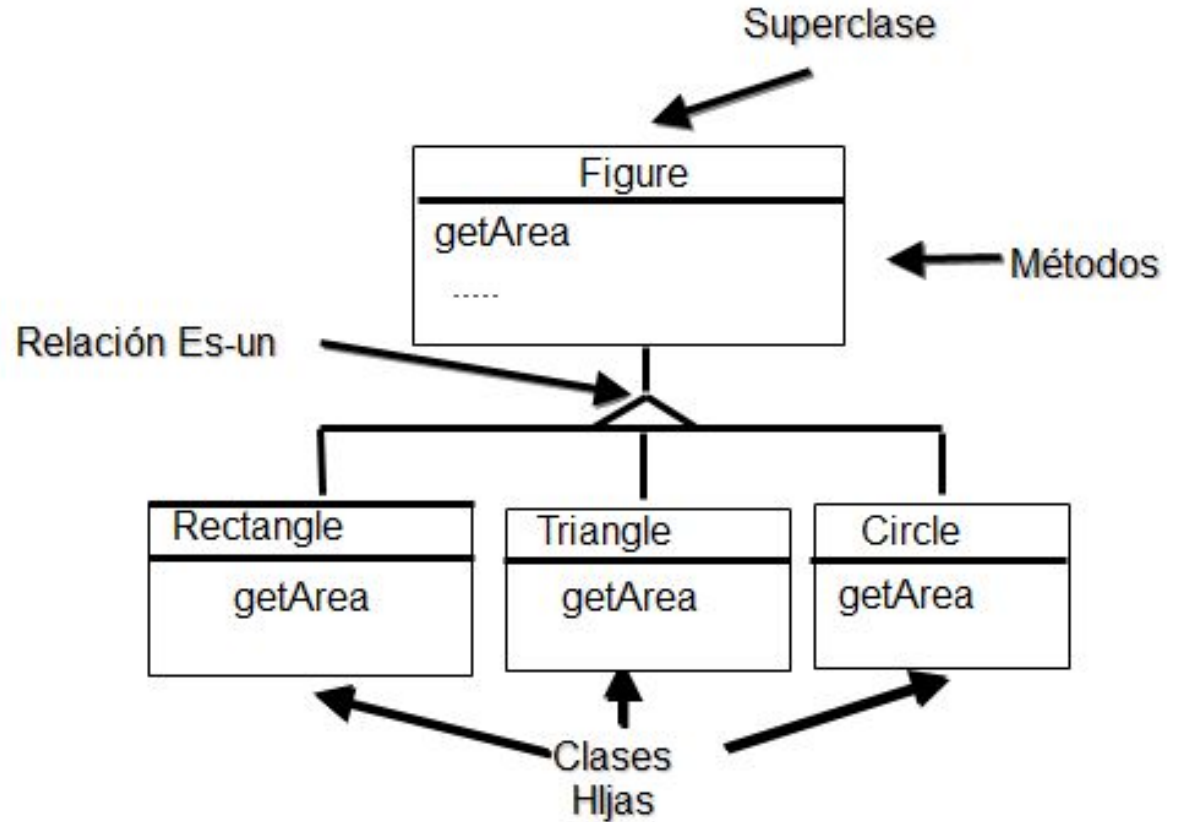
- Es posible abstraer características comunes de varias clases en una “superclase”
- EL mecanismo de abstracción sirve como mecanismo de clasificación de entidades
- La extensión permite ampliar las características de una clase en una subclase
- Es un mecanismo de reuso tanto a nivel de diseño como implementación

# Que tienen en común Triangulo, Circulo y Rectangulo?

— — —



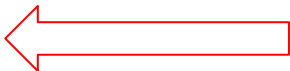
# Herencia: Ejemplo



# 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**



---